## Contents

About this book ................................................................................................. v

MobiLink server technology .............................................................................. 1
  - MobiLink consolidated databases .......................................................... 1
  - MobiLink server ....................................................................................... 27
  - MobiLink server options ......................................................................... 40
  - Synchronization techniques ..................................................................... 98
  - MobiLink performance .......................................................................... 131
  - Central administration of remote databases .......................................... 140
  - MobiLink Profiler .................................................................................. 176
  - The Relay Server .................................................................................. 195
  - MobiLink file-based download ............................................................... 195

MobiLink events .............................................................................................. 211
  - Synchronization script writing ............................................................. 211
  - Synchronization events .......................................................................... 248

MobiLink server APIs ..................................................................................... 427
  - Synchronization script writing in Java .................................................. 427
  - MobiLink server Java API reference ..................................................... 440
  - Synchronization scripts in .NET ............................................................. 497
  - MobiLink server .NET API reference .................................................... 512
  - Direct row handling .............................................................................. 585

MobiLink reference ......................................................................................... 597
  - MobiLink Replay C++ callbacks ............................................................ 597
  - MobiLink server system procedures ..................................................... 605
  - MobiLink utilities .................................................................................. 652
  - MobiLink data mappings between remote and consolidated databases ... 666
  - Character set considerations ................................................................. 712
ODBC drivers for MobiLink ................................................................. 714
MobiLink application deployment ......................................................... 717

Index .................................................................................................. 733
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.
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
- SAP HANA
- SAP Sybase IQ
- 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 “Consolidated database setup” on page 2.

For information about writing synchronization scripts for particular consolidated databases, see “RDBMS-dependent synchronization scripts” on page 4.

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 585.

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 3.

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 666

Consolidated database setup

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 %SQLANY16%\MobiLink\setup. For more information about MobiLink system setup, see “MobiLink system setup” [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 605.

**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 user must be used to start the MobiLink server and to configure MobiLink. See “Required privileges” on page 28.

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

- “Adaptive Server Enterprise consolidated database”
- “IBM DB2 LUW consolidated database”
- “Microsoft SQL Server consolidated database”
- “MySQL consolidated database”
- “Oracle consolidated database”
- “SAP HANA consolidated database” on page 23
- “SAP Sybase IQ consolidated database”
- “SQL Anywhere consolidated database”

**ODBC connection**
The MobiLink server needs an ODBC connection to your consolidated database. You must install 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 “ODBC drivers for MobiLink” on page 714.

For updated information and complete functional specifications of the ODBC drivers you can use with MobiLink, see [http://www.sybase.com/detail?id=1011880](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 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 database 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 “Consolidated database setup” 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 “Synchronization script writing” on page 211.

For a complete list of events you can write scripts for, see “Synchronization events” on page 248.
For specific information about each type of consolidated database, see:

- “SQL Anywhere consolidated database”
- “Adaptive Server Enterprise consolidated database”
- “IBM DB2 LUW consolidated database”
- “Microsoft SQL Server consolidated database”
- “MySQL consolidated database”
- “Oracle consolidated database”
- “SAP HANA consolidated database” on page 23
- “SAP Sybase IQ consolidated database”

.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 “Java synchronization logic” on page 429.

For information about .NET synchronization logic, see “Synchronization scripts in .NET” on page 497.

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 mlsrv16 -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 mlsrv16 option” on page 46.

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 666.
Synchronization of 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 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}
```
The first named parameter represents the WKB value and the second named parameter is for the SRID.

**Oracle considerations**

When downloading spatial data from Oracle, keep the following in mind:

- Oracle server does not generate Well-Known-Binary (WKB) values if a geometry value is entered into an Oracle database using the Oracle native format. The following example shows a work-around.

For data entered in the native Oracle spatial format, like this:

```sql
CREATE TABLE cola_markets (
    mkt_id NUMBER PRIMARY KEY,
    name VARCHAR2(32),
    shape SDO_GEOMETRY);

INSERT INTO cola_markets VALUES(1, 'cola_a', SDO_GEOMETRY(2003, -- two-dimensional polygon NULL, NULL, SDO_ELEM_INFO_ARRAY(1,1003,3), -- one rectangle (1003 = exterior) SDO_ORDINATE_ARRAY(1,1, 5,7) -- only 2 points needed to define rectangle (lower left and upper right) with Cartesian-coordinate data ));
```

The download cursor must be written as follows or the default download cursor returns 1 instead of the actual binary value:

```sql
COLA_MARKETS (SCOTT): download_cursor
SELECT t.MKT_ID, t.NAME,
    SDO_UTIL.TO_WKBGEOMETRY(SDO_UTIL.FROM_WKTGEOMETRY(SDO_UTIL.TO_WKTGEOMETRY(t.SHAPE))),
    t.SHAPE.sdo_srid
FROM SCOTT.COLA_MARKETS t
```

- The default download script works correctly if the data has been entered in WKT format, as follows:

```sql
INSERT INTO cola_markets VALUES(1, 'cola_a', SDO_GEOMETRY( 'polygon (( 1 1, 5 1, 5 7, 1 7, 1 1))',4326));
INSERT INTO SCOTT.cola_markets VALUES(2, 'cola_b', SDO_GEOMETRY( 'polygon (( 5 1, 8 1, 8 6, 5 7, 5 1))',4326));
INSERT INTO SCOTT.cola_markets VALUES(3, 'cola_c', SDO_GEOMETRY( 'polygon (( 3 3, 6 3, 6 5, 4 5, 3 3))',4326));
```

- Oracle does not have a SRID of 0, so all flat Cartesian plane data should be entered using a SRID of 4326.
See also

- “Spatial data” [SQL Anywhere Server - Spatial Data Support]
- “Named script parameters” on page 216
- “User-defined named parameters” on page 230

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:

\[
\text{create table test (c1 int not null primary key, c2 st\_geometry )}
\]

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

- “SQL Anywhere consolidated database”
- “Microsoft SQL Server consolidated database”
- “MySQL consolidated database”
- “Oracle consolidated database”
- “SAP Sybase IQ consolidated database”

SQL Anywhere sample scripts

The following is a sample upload\_insert script for SQL Anywhere:

\[
\text{INSERT INTO test VALUES( \{ml r.c1\}, ST\_Geometry::ST\_GeomFromBinary(\{ml r.c2\:data\},\{ml r.c2\:srid\}) )}
\]

The following is a sample download\_cursor script for SQL Anywhere:

\[
\text{SELECT c1, c2.ST\_AsBinary(), c2.ST\_SRID() FROM test}
\]

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:
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
```

Adaptive Server Enterprise consolidated database

Prerequisites

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 privileges” on page 28.

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 privilege on MASTER..SYSTRANSACTIONS.

The MobiLink server login ID must have the dtm_tm_role role, if the -cs option for mlsrv16 is used.

You must use the sp_dboption option to set the SELECT INTO option to true. For example, run the following script in Interactive SQL to set the SELECT INTO privilege to true on your-database-name:

```
sp_dboption your-database-name, "SELECT INTO", true
```

go

**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 %SQLANY16%\MobiLink\Setup.
- Check and update the MobiLink system setup from Sybase Central. See “MobiLink system setup” [MobiLink - Getting Started].

**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:

- Your Adaptive Server Enterprise documentation

**Adaptive Server Enterprise considerations**

- **Enable functionality group configuration parameter** When the enable functionality group configuration parameter is enabled on the ASE 15.7 server, the MobiLink server uses the "select ... for update" feature to lock remote IDs to prevent redundant synchronizations for the same remote ID simultaneously. If you turn off the enable functionality group parameter, you must restart any MobiLink servers currently connected to the ASE server to avoid failure of synchronization requests.

- **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 666.

- **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 mlsrv16 -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 mlsrv16 option” on page 46.

- **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 129.

**IBM DB2 LUW consolidated database**

MobiLink supports IBM DB2 LUW for Linux and Windows.

**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.

**Prerequisites**

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 privileges” on page 28.

- 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.

**Context and remarks**

You can also check and update the MobiLink system setup from Sybase Central. See “MobiLink system setup” [MobiLink - Getting Started].
Task

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
   ```

Results

MobiLink system tables, stored procedures, triggers, and views are installed and IBM DB2 LUW can be used as a consolidated database.

Next

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:

- IBM DB2 LUW documentation

IBM DB2 LUW considerations

- **Lock escalation**  
  To maintain data consistency between the consolidated and remote databases, the MobiLink server issues the following query through the `ml_lock_rid` stored procedure to lock the remote ID in every synchronization phase.

  ```
  SELECT sync_key into p_sync_key FROM ml_database WHERE rid = a_given_remote_id WITH RR USE AND KEEP EXCLUSIVE LOCKS;
  ```

  This query exclusively locks the remote ID to prevent any concurrent synchronizations using the same remote ID.

  If you experience any MobiLink remote ID locking errors when there are no concurrent synchronizations for the same remote ID, for example, if MobiLink error code -10341 is in the
MobiLink server log, you may need to adjust the DB2 maxlocks and locklist configuration parameters to prevent lock escalation. Consult the DB2 documentation for full details.

- **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 675.

- **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 mlsrv16 -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 mlsrv16 option” on page 46.

- **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:

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

  For an example using the sample application, see “CustDB sample for MobiLink” [MobiLink - Getting Started].

- **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 129.

## Microsoft SQL Server consolidated database

### Prerequisites

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 privileges” on page 28.
● 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:

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

● The MobiLink server login ID must have SELECT privilege on sys.databases SYS.DM_tran_LOCKS, SYS.PARTITIONS, and SYS.SYSProcesses.

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 `%SQLANY16%\MobiLink\Setup`.

● Check and update the MobiLink system setup from Sybase Central. See “MobiLink system setup” [MobiLink - Getting Started].

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:


● 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 683.

● **BLOB columns** Due to restrictions of the SQL Server ODBC driver, all BLOB columns should be placed at the end of columns when a synchronization table is defined, especially when the download_cursor script for the table must be written as a stored procedure call or a batch of SQL statements. This restriction can be ignored if the download_cursor script for a synchronization table that contains BLOB columns is written as a single SELECT statement. See [http://support.microsoft.com/kb/238080](http://support.microsoft.com/kb/238080).

● **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 mlsrv16 -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 mlsrv16 option” on page 46.
● **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:

```sql
( 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**  For SQL Server, you must perform conflict detection and resolution in the upload_update script. See “Conflict resolution with upload_update scripts” on page 120.

**Isolation level**

See “MobiLink isolation levels” on page 129.

---

**MySQL consolidated database**

The MobiLink server supports MySQL Community and Enterprise servers 5.1.3 or later.

**Privileges**

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 privileges” on page 28.

● 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.

**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. Failure to do so may cause data inconsistencies.

If necessary, before applying the file against your MySQL database, edit the `syncmys.sql` script file to add the following two lines after the line `delimiter //`, where `engine_name` is an ACID compliant storage engine.

```sql
set storage_engine = [engine_name]
//
```
Setting up MySQL as a consolidated database using the setup script

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.

Prerequisites

Your MySQL user ID must have privileges to create tables, procedures, functions, views, and triggers.

Context and remarks

You can also use the MobiLink 16 plug-in for Sybase Central to set up MySQL as a consolidated database.

Task

- Using the MySQL command line tool or the MySQL Query Browser, run the `syncmys.sql` setup script, located in `%SQLANY16%MobiLink\Setup`.

Results

The MySQL database is ready to be used as a MobiLink consolidated database.

Setting up MySQL as a consolidated database using Sybase Central

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.

Prerequisites

Your MySQL user ID must have privileges to create tables, procedures, functions, views, and triggers.

Context and remarks

You can also use the `syncmys.sql` setup script to set up MySQL as a consolidated database.

Task

1. In the MobiLink 16 plug-in for Sybase Central, choose Folders from the View menu.
2. Open your MobiLink project and expand Consolidated Databases.
3. Right-click the database name and click Check MobiLink System Setup. If your database requires setup, you are prompted to continue.

   To use an existing MobiLink system setup, your default_schema should be the schema of the MobiLink system setup.

Results

The MySQL database is ready to be used as a MobiLink consolidated database.
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:


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 690.

- **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 correctly.

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

- **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 129.

Oracle consolidated database

Privileges

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 privileges” on page 28.

- 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).

The RDBMS user must also have SELECT privilege on GV$TRANSACTION, GV$SESSION, GV$LOCK, and DBA_OBJECTS, and EXECUTE privileges on DBMS_UTILITY. You cannot grant permission directly for the GV$TRANSACTION, GV$SESSION and GV$LOCK synonyms; you must instead grant permission on the underlying GV_$TRANSACTION, GV_$SESSION, and GV_$LOCK dynamic performance views. You must connect as SYS to grant this access. The Oracle syntax for granting this access is:

```
grant select on SYS.GV_$TRANSACTION to user-name;
grant select on SYS.GV_$SESSION to user-name;
grant select on SYS.GV_$LOCK 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 %SQLANY16%\MobiLink\Setup.

- Check and update the MobiLink system setup from Sybase Central. See “MobiLink system setup” [MobiLink - Getting Started].
ODBC driver

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

- “SQL Anywhere 16 - Oracle ODBC driver”

Oracle considerations

- **MobiLink synchronization and timestamp-based downloads with an Oracle Real Application Cluster**  
  Rows in the consolidated database running on an Oracle RAC may be missed if the clocks of the Oracle cluster nodes differ by more than the time elapsed between the MobiLink server fetching the next last download timestamp and fetching the rows to be downloaded. This problem is unlikely on a RAC system with synchronized node clocks, but the likelihood increases with larger node clock differences. A workaround is to create either a modify_next_last_download_timestamp or modify_last_download_timestamp script to subtract the maximum node clock difference.

At least since version 10i, Oracle has recommended using Network Time Protocol (NTP) to synchronize the clocks on all nodes in a cluster. NTP typically runs by default on Unix and Linux. With cluster nodes properly configured to use NTP, their clocks should all be within 200 microseconds to 10 milliseconds (depending on the proximity of the NTP server). Since Windows Server 2003, the Windows Time Service implements the NTP version 3 protocol which runs by default. Also, as of version 11gR2, Oracle Clusterware includes the Oracle Cluster Time Synchronization Service (CTSS) to either monitor clock synchronization or, if neither NTP or Windows Time Service is running, to actively maintain clock synchronization. However, CTSS and Windows Time Service are less accurate than NTP.

To avoid missing rows when Oracle RAC node clocks differ by up to one second more than the time between fetching the next_last_download_timestamp and the rows to be downloaded, the MobiLink server subtracts one second from the next_last_download_timestamp fetched from the consolidated database if the following are true:

- the Oracle account used by the MobiLink server has execute privilege for SYS.DBMS_UTILITY
- the consolidated database is an Oracle RAC system
- for MobiLink versions 12.0.0 and up, there is no generate_next_last_download_timestamp script

For Oracle RAC node clocks that may differ by greater amounts, you can avoid the problem by defining a generate_next_last_download_timestamp, modify_next_last_download_timestamp or modify_last_download_timestamp script to compensate for the maximum node clock difference.

- **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 695.

- **XMLTYPE data type**  
  Use of the Oracle XMLTYPE data type with SQL Anywhere or UltraLite requires special care. For details, see “Oracle XMLTYPE data type” on page 20.

- **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 mlsrv16 -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 mlsrv16 option” on page 46.

- **Timestamps**  The MobiLink server uses gv$transaction to generate a timestamp for the remote database to be used in the next synchronization, so the MobiLink server login ID must have a SELECT privilege on gv$transaction. Oracle does not allow you to grant access to gv$transaction directly; you must instead grant SELECT privilege on the underlying gv_$transaction view. See “Privileges” on page 18.

- **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 SQL Anywhere 16 - Oracle ODBC driver. Also, Sybase Central requires procedures to return results to use central administration of remote databases, so this option needs to be selected when using central administration.

See “SQL Anywhere 16 - Oracle ODBC driver” on page 714.

- **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.

**Oracle XMLTYPE data type**

The Oracle XMLTYPE data type can be mapped to the SQL Anywhere XML data type or the UltraLite LONG VARCHAR or VARCHAR(n) data types. It is important to be aware that the Oracle database server validates the data before storing it into an XMLTYPE column but SQL Anywhere and UltraLite do not, so you must ensure that XML documents to be uploaded contain valid XML.

Small XML documents with a length of less than or equal to 32 KB can be uploaded into and downloaded from an Oracle database with Oracle PL/SQL statements. When the length of XML documents is greater than 32 KB, the upload XML documents may need to be uploaded into a global temporary table using the upload_insert and upload_update scripts. The upload data can then be converted and stored into the actual synchronization table using the end_upload_rows or end_upload script.
The following examples provide sample upload and download scripts to upload and download XMLTYPE objects between an Oracle consolidated database and SQL Anywhere remote databases. In these examples, the upload table is defined in the Oracle consolidated database as:

```sql
create table test (pk int not null primary key, c1 XMLTYPE)
```

The upload table is defined in the SQL Anywhere remote database as:

```sql
create table test (pk int not null primary key, c1 XML)
```

● When all XML documents are less than or equal to 32KB long, the upload and download scripts can be written as follows

```sql
upload_insert

declare v_pk integer; v_c1 clob; x_c1 xmltype;
begin
  v_pk := {ml r.pk};
  v_c1 := {ml r.c1};
  x_c1 := XMLTYPE.createXML( v_c1 );
  insert into test values( v_pk, x_c1 );
end;

download_cursor

select pk, XMLSERIALIZE( content c1 ) from test
```

This upload_insert script works well when the XML data length is less than or equal to 32 KB. However, if the XML data length is greater than 32 KB, the Oracle server may issue an error.

● If there are any XML documents greater than 32 KB long, the upload XML data needs to be uploaded in a global temporary table

The upload_insert script uploads the XML documents into a global temporary table in the Oracle consolidated database. The global temporary table is defined as:

```sql
create global temporary table tmp_test (pk int, c1 CLOB)
```

Then the upload_insert script can written as follows:

```sql
insert into tmp_test values( {ml r.pk}, {ml r.c1} )
```

Note
The c1 column in the temporary table must have the CLOB data type.

The end_upload_rows script retrieves the XML documents from the global temporary table, converts them to XML documents, and then stores the XML data into the test table. Following is the end_upload_rows script:

```sql
insert into test (pk, c1) (select pk, XMLTYPE.createXML(c1) from tmp_test
```

Isolation level
See “MobiLink isolation levels” on page 129.
Oracle VARRAY

The SQL Anywhere 16 - 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 cleared.
- 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 mlsrv16 option" on page 71.
SAP HANA consolidated database

Privileges
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 access the MobiLink system tables during synchronization. This user must be used to start the MobiLink server and to configure MobiLink applications. See “Required privileges” on page 28.

- 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 RDBMS user must also have the CATALOG READ privilege.

Setting up SAP HANA as a consolidated database
To set up SAP HANA to work as a MobiLink consolidated database, you must run a setup procedure that adds MobiLink system tables, stored procedures, and views that are required for MobiLink synchronization. Use the following method to set up SAP HANA.

- Run the synchana.sql setup script with SAP HANA Studio. The script file is located in %SQLANY16%\MobiLink\Setup.

ODBC driver
The MobiLink server can only recognize the ODBC driver from SAP HDBODBC for SAP HANA databases. See:

- Your SAP HANA documentation

Stored procedures
SAP HANA does not support calling stored procedures with NULLs for scalar input parameters, so you should not implement the upload_insert and upload_update scripts as stored procedure calls for any synchronization tables with nullable columns. The MobiLink server replaces NULLs with zero length string or binary values for parameters with CHAR or BINARY types respectively, and then passes them to the SAP HANA database when it calls the user authentication scripts. The user authentication stored procedures should be written with this MobiLink server behavior in mind.

The user-defined parameters are also set to zero length strings when first referenced.

SAP HANA considerations
- MobiLink System Database (MLSD) The MobiLink server does not support MLSD for SAP HANA.

- MobiLink server system objects The primary key columns in the MobiLink server system tables are maintained by SAP HANA SEQUENCEs through stored procedure calls.
Use the `ml_add_connection_script` and `ml_add_table_script` system procedures to add, modify, or delete connection and table scripts, and the `ml_add_user` and `ml_add_database` system procedures to add MobiLink users and remote databases. Do not directly insert any rows into the MobiLink server system tables.

See:
- “`ml_add_connection_script system procedure`” on page 609
- “`ml_add_table_script system procedure`” on page 624
- “`ml_add_user system procedure`” on page 626

**Timestamp-based downloads**  Because SAP HANA supports snapshot isolation, the MobiLink server uses the start time of the oldest open transaction under the current user as the next `last_download_timestamp` so that the `download_cursor` and `download_delete_cursor` scripts can use the last modified column information to generate a timestamp-based download stream. Following is an example.

1. Create a synchronization table in SAP HANA.

   ```sql
   create COLUMN table test (pk int primary key, cl int, last_modified timestamp generated always as ( now() ) )
   ```

   The clause `generated always as ( now() )` causes the SAP HANA server to update the `last_modified` column with the current timestamp, whenever this row is inserted or updated.

2. The `download_cursor` script can be written as follows.

   ```sql
   select pk, cl from test where last_modified > {mls.last_table_download}
   ```

   SAP HANA does not support triggers, so you may have to use logical deletes to generate download deletes from SAP HANA to the remote databases.

Alternatively, the `download_cursor` and `download_delete_cursor` scripts can be written based on the hidden columns `$validto$` and `$validfrom$` when the table is created with the `HISTORY COLUMN` clause and the next `last_download_timestamp` can be generated by the `generate_next_last_download_timestamp` script. For information about these hidden columns, see the SAP HANA documentation.

If the `download_cursor` and/or `download_delete_cursor` scripts access any tables owned by database users other than the user used by the MobiLink server to login to the SAP HANA database, you need to grant `CATALOG READ` privilege to the MobiLink server login user using the following statement:

   ```sql
   GRANT CATALOG READ TO MobiLink_server_login_user_name
   ```

This enables the MobiLink server login user to see all the open transactions in the database, and not just open transactions for the current user, when trying to get the oldest open transaction of the database.

**Data type mapping**  The data types of columns must map correctly between your consolidated and remote database. See “SAP HANA database server data type mapping” on page 704.
Isolation level
   See “MobiLink isolation levels” on page 129.

SQL Anywhere consolidated database

Privileges
   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 privileges” on page 28.

   ● 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 %SQLANY16%MobiLink\Setup.

   ● Check and update the MobiLink system setup from Sybase Central. See “MobiLink system setup”
     [MobiLink - Getting Started].

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 “ODBC data sources” [SQL Anywhere
   Server - Database Administration].

Isolation level
   See “MobiLink isolation levels” on page 129.

SAP Sybase IQ consolidated database
   [ This topic has been updated for build 1823. ]

Privileges
   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 privileges” on page 28.
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 EXECUTE privilege on the SP_IQTRANSACTION system procedure for SAP Sybase IQ.

Setting up SAP Sybase IQ as a consolidated database

To set up SAP Sybase IQ 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 synciq.sql setup script, located in %SQLANY16%MobiLink\Setup.
- Check and update the MobiLink system setup from Sybase Central. See “MobiLink system setup” [MobiLink - Getting Started].

Setting up the ODBC driver

You must set up an ODBC DSN for your SAP Sybase IQ consolidated database. The ODBC driver for SAP Sybase IQ is installed with Sybase IQ.

For information about the ODBC driver for SAP Sybase IQ, see the SAP Sybase IQ documentation.

SAP Sybase IQ considerations

With row-level versioning

- With the SAP Sybase IQ 16 row-level versioning (RLV) feature more than one user can modify the same table concurrently, users can wait for transaction locks instead of having to retry, and a hybrid storage model optimizes data write-access, without sacrificing read-access performance. In order to get better upload performance, we recommend that all synchronization tables be RLV enabled.

Without row-level versioning or if you are using SAP Sybase IQ 15.4

- If uploads contain upload data that modifies any synchronization tables that were defined on the SAP Sybase IQ store, and if the MobiLink server is running with more than one concurrent database worker thread, all the uploads must be serialized because SAP Sybase IQ server allows only a single connection to modify a given table on the SAP Sybase IQ store at any given time.

This requirement can be achieved, if the begin_upload connection script is written to include or to use the following SQL statement:

\[\text{LOCK TABLE table\_name IN WRITE MODE WAIT time\_string}\]

where table\_name is the name of a table that is defined on the SAP Sybase IQ store and the time\_string gives the maximum time period to lock the table. The table can be as simple as one defined as follows:

\[\text{create table coordinate\_upload ( c1 int )}\]

The table is not required to have any data.
All transactions that modify SAP Sybase IQ tables must be serialized, whether they occur on MobiLink server connections or on other connections to the SAP Sybase IQ database. For MobiLink server transactions, the same logic outlined above can be used. This technique is more efficient than having the MobiLink server automatically retry on each transaction and results in better performance.

When creating a synchronization model for an SAP Sybase IQ consolidated database, the table mappings default to download-only for SAP Sybase IQ tables. If you change any mappings to bi-directional or upload-only, you must ensure that changes to those SAP Sybase IQ tables are serialized. For example, by adding a begin_upload event as described above.

Isolation level
See “MobiLink isolation levels” on page 129.

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.

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.

Note
The mlsrv16 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 248.

To start the MobiLink server, run mlsrv16. Use the -c option to specify the ODBC connection parameters for your consolidated database. For a list of mlsrv16 command line options, see “MobiLink server options” on page 40.

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 information about connection parameters, see “-c mlsrv16 option” on page 48.

The MobiLink server needs an ODBC Data Source Name (DSN) to communicate with the consolidated database. A DSN includes information for the ODBC Driver Manager on where to load the ODBC driver. On Windows, ODBC data sources can be created with the Microsoft ODBC Data Source Administrator. The bitness of the MobiLink server must match the bitness of the DSN. More precisely, a 64-bit MobiLink server must use a 64-bit DSN created via ODBC Data Source Administrator (64-bit). See “Lesson 1: Setting up a MobiLink consolidated database” [MobiLink - Getting Started].

Example
The following command starts the MobiLink server using the ODBC data source SQL Anywhere 16 CustDB to identify the consolidated database. Enter the entire command on one line.

```mlsrv16 -c "DSN=SQL Anywhere 16 CustDB;UID=ml_server;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 mlsrv16 option” on page 48
- “-zs mlsrv16 option” on page 95
- “-o mlsrv16 option” on page 61
- “-v mlsrv16 option” on page 79
- “-x mlsrv16 option” on page 85

You can also start the MobiLink server as a Windows service or Unix daemon. See “MobiLink server use outside the current session” on page 31.

**Required privileges**

You must specify a database user for the MobiLink server to connect to the database server. You specify the database user with the mlsrv16 -c option or in the ODBC data source. See “-c mlsrv16 option” on page 48.

This database user must have full SELECT, INSERT, UPDATE, and DELETE privileges on the MobiLink system tables, and must also have the EXECUTE ANY PROCEDURE privilege on the MobiLink system procedures. By default, the database user who runs the MobiLink setup script has these privileges. To use another database user to run the MobiLink server, you must grant these privileges for that user on the ml_* tables and the ml_add_*_script system procedures.

For a list of all MobiLink system procedures, see “MobiLink server system procedures” on page 605.

The database user also needs the appropriate privilege on all tables referenced in the MobiLink scripts, and EXECUTE privileges on any procedures referenced in the MobiLink scripts.

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

- “Adaptive Server Enterprise consolidated database”
- “IBM DB2 LUW consolidated database”
- “Microsoft SQL Server consolidated database”
- “MySQL consolidated database”
- “Oracle consolidated database”

For more information about setup scripts, see “Consolidated database setup” on page 2.

**MobiLink connectivity**

When using HTTP or HTTPS, with or without the Relay Server, you can use a web browser to verify MobiLink server is listening for requests. For example, if your MobiLink server command line is as follows:

```
mlsrv16 ... -x http(port=8080)
```
and the computer is *ml1.mycorp.com*, then you can open a web browser and point it to `http://ml1.mycorp.com:8080`.

MobiLink server responds with a **404 Not Found** error that also mentions the MobiLink server’s major version.

See “Introduction to the Relay Server” [Relay Server]

**MobiLink server shutdown**

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).
- Click **Shut down** on the MobiLink server messages window.
- In Windows, right-click the MobiLink server icon in the system tray and click **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

- “MobiLink Stop utility (mlstop)” on page 652
- “ServerContext.shutdown method [MobiLink server Java]” on page 475
- “ServerContext.Shutdown method [MobiLink server .NET]” on page 565

**MobiLink server logging**

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`.

```
mlsrv16 -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 mlsrv16 option” on page 61
- “-on mlsrv16 option” on page 61
- “-os mlsrv16 option” on page 62
- “-ot mlsrv16 option” on page 63

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 mlsrv16 option” on page 79.

Controlling which warning messages are reported
You can control which warning messages are reported.

For more information, see:

- “-zw mlsrv16 option” on page 96
- “-zwd mlsrv16 option” on page 97
- “-zwe mlsrv16 option” on page 97

MobiLink server log viewing
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 29.

MobiLink Server Log File Viewer
To view MobiLink server logs, open Sybase Central and click Tools » MobiLink 16 » MobiLink Server Log File Viewer. You are prompted to choose a log file to view.

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.
MobiLink server use 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 mlsrv16 -ud option, enabling the MobiLink server to run in the background, and to continue running after you log off.
- **Windows service** You can run the Windows MobiLink server as a service.

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

See also
- “-ud mlsrv16 option” on page 78
- “MobiLink server as a service on Windows” on page 31
- “MobiLink Stop utility (mlstop)” on page 652
- “Service utility (dbsvc) for Linux” [SQL Anywhere Server - Database Administration]
- “How to run the database server as a service or daemon” [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.

**Prerequisites**

There are no prerequisites for this task.

**Task**

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

  ```
  mlsrv16 -c "DSN=SQL Anywhere 16 Demo;UID=DBA;PWD=sql" -ud
  ```

**Results**

The Unix MobiLink server as running as a daemon

See also
- “-ud mlsrv16 option” on page 78
- “Service utility (dbsvc) for Linux” [SQL Anywhere Server - Database Administration]

MobiLink server as a service on Windows

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]

Working with services

Use Sybase Central to add a new service, or modify or delete an existing service. Changes to a service configuration take effect the next time the service is started.

Prerequisites

There are no prerequisites for performing this task.

Context and remarks

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.

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

Task

1. In Sybase Central, in the left pane, click MobiLink 16.
2. In the right pane, click the Services tab.
3. In the right pane, right-click and click New » Service.
   
   To delete a service, choose the service and then click Edit » Delete.
   
   To change the parameters for a service, right-click the service and click Properties.
4. Follow the instructions in the Create Service Wizard.

Results

The service is added, deleted or modified.
Startup options for services

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.

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 16 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 40.

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 task bar 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/2003 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 cleared 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

You can change the program executable file associated with a service in Sybase Central.

### Prerequisites

There are no prerequisites for this task.

### Context and remarks

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

### Task

1. Click the **Configuration** tab on the **Service Properties** window.
2. Type the new path and file name in the **File Name** box.

### Results

The program executable file associated with the service is updated.

## Starting and stopping a service

### Starting and stopping a service

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

### Prerequisites

There are no prerequisites for performing this task.

### Context and remarks

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

### Start or stop a service

1. In Sybase Central, click MobiLink 16 in the left pane, and then open the **Services** tab in the right pane.
2. Right-click the service and click **Start** or **Stop**.

### Results

The service is started or stopped.
Multiple services

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.

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. See “Adding a service or group to a list of dependencies” on page 36.

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. See “Checking and changing which group a service belongs to” on page 35.

Checking and changing which group a service belongs to

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.

Prerequisites

There are no prerequisites for this task.

Task

1. In Sybase Central, click MobiLink 16 in the left pane, and then open the Services tab in the right pane.
2. Right-click the service and click 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.

**Results**

Result

**Next**

You can configure your services to ensure they start in the correct order. See “Adding a service or group to a list of dependencies” on page 36.

**Adding a service or group to a list of 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.

**Prerequisites**

There are no prerequisites for this task.

**Task**

1. In Sybase Central, click MobiLink 16 in the left pane, and then open the Services tab in the right pane.

2. Right-click the service and click 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.

**Results**

The service or group is added to the list of dependencies.

**MobiLink server in a server farm**

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
</table>

**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 16 - 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.

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:

```
mlsrv16 -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, 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 also

- “MobiLink Arbiter Server utility for Windows (mlarbiter)” on page 663
- “-lsc mlsrv16 option” on page 59
- “-ca mlsrv16 option” on page 48

**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 mlsrv16 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 mlsrv16 option” on page 50
- “-cmin mlsrv16 option” on page 50
- “-cinit mlsrv16 option” on page 49

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 for MobiLink server resources” [SQL Anywhere Server - Database Administration].

- **Percent of Pages Locked**  
  This is a SQL Anywhere Monitor metric that gives the percentage of pages in the cache that cannot be transferred to disk. See “List of metrics for MobiLink server resources” [SQL Anywhere Server - Database Administration].

- **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 for MobiLink server resources” [SQL Anywhere Server - Database Administration].

- **PAGES_LOCKED_MAX**  
  The -ppv option for mlsrv16 can print the number of pages in the memory cache. See “-ppv mlsrv16 option” on page 63.

- **PAGES_LOCKED**  
  The -ppv option for mlsrv16 can print the number of cache pages loaded into memory. See “-ppv mlsrv16 option” on page 63.

- **PAGES_USED**  
  The -ppv option for mlsrv16 can print the number of cache pages used. See “-ppv mlsrv16 option” on page 63.

- **-vk option for mlsrv16**  
  Causes the server to print a line to the log when the cache grows or shrinks. See “-ppv mlsrv16 option” on page 63.

**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 options 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.
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 mlsrv16. It is worth noting that non-SQL Anywhere ODBC drivers can use a significant amount of memory, particularly when processing BLOB columns.

See:
- “-nc mlsrv16 option” on page 59
- “-sm mlsrv16 option” on page 74
- “-w mlsrv16 option” on page 83

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 “MobiLink server logging” on page 29.

Verify MobiLink server connectivity
When using HTTP or HTTPS, with or without the Relay Server, you can use a web browser to verify MobiLink server is listening for requests. For example, if your MobiLink server command line is as follows:

mlsrv16 ... -x http(port=8080)
and the computer is ml1.mycorp.com, then you can open a web browser and point it to http://ml1.mycorp.com:8080.

MobiLink server responds with a 404 Not Found error that also mentions the MobiLink server's major version.

See “Introduction to the Relay Server” [Relay Server].

**MobiLink server options**

**mlsrv16 syntax**

Starts a MobiLink server.

**Syntax**

`mlsrv16 -c "connection-string" [ options ]`

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@data</td>
<td>Read in options from the specified environment variable or configuration file. See “@data mlsrv16 option” on page 45.</td>
</tr>
<tr>
<td>-a</td>
<td>Keep using a consolidated database connection after a synchronization error on that connection. See “-a mlsrv16 option” on page 46.</td>
</tr>
<tr>
<td>-b</td>
<td>Trim blank padding of strings. See “-b mlsrv16 option” on page 46.</td>
</tr>
<tr>
<td>-bn size</td>
<td>Specify the maximum number of bytes to consider when comparing BLOBs for conflict detection. See “-bn mlsrv16 option” on page 47.</td>
</tr>
<tr>
<td>-c &quot;keyword=value; ...&quot;</td>
<td>Supply ODBC database connection parameters for your consolidated database. This option is required. See “-c mlsrv16 option” on page 48.</td>
</tr>
<tr>
<td>-ca host_or_ip</td>
<td>Set the host name or IP address for the MobiLink arbiter server. See “-ca mlsrv16 option” on page 48.</td>
</tr>
<tr>
<td>-cinit size</td>
<td>Specify the initial size for the server memory cache. See “-cinit mlsrv16 option” on page 49.</td>
</tr>
<tr>
<td>-cm size</td>
<td>Specify the server memory cache size. See “-cm mlsrv16 option” on page 49.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-cmax size</td>
<td>Specify the maximum size for the server memory cache. See “-cmax mlsrv16 option” on page 50.</td>
</tr>
<tr>
<td>-cmin size</td>
<td>Specify the minimum size for the server memory cache. See “-cmin mlsrv16 option” on page 50.</td>
</tr>
<tr>
<td>-cn connections</td>
<td>Set the maximum number of simultaneous connections with the consolidated database server. See “-cn mlsrv16 option” on page 50.</td>
</tr>
<tr>
<td>-cr count</td>
<td>Set the maximum number of database connection retries. See “-cr mlsrv16 option” on page 52.</td>
</tr>
<tr>
<td>-cs &quot;keyword=value; ...&quot;</td>
<td>Supply system database connection parameters for your MobiLink System Database (MLSD). See “-cs mlsrv16 option” on page 52.</td>
</tr>
<tr>
<td>-ct connection-timeout</td>
<td>Set the length of time a connection may be unused before it is timed out. See “-ct mlsrv16 option” on page 53.</td>
</tr>
<tr>
<td>-dl</td>
<td>Display all log messages in the MobiLink server messages window. See “-dl mlsrv16 option” on page 53.</td>
</tr>
<tr>
<td>-dr</td>
<td>For Adaptive Server Enterprise only. Ensures that tables involved in synchronization do not use the DataRow locking scheme. See “-dr mlsrv16 option” on page 53.</td>
</tr>
<tr>
<td>-ds size</td>
<td>Specify the maximum amount of data that can be stored for use in all restartable downloads. See “-ds mlsrv16 option” on page 54.</td>
</tr>
<tr>
<td>-dsd</td>
<td>Disable snapshot isolation, which is the default download isolation level for SQL Anywhere and Microsoft SQL Server consolidated databases. See “-dsd mlsrv16 option” on page 54.</td>
</tr>
<tr>
<td>-dt</td>
<td>Detect transactions only within the current database. See “-dt mlsrv16 option” on page 55.</td>
</tr>
<tr>
<td>-e filename</td>
<td>Store remote database error logs sent into the named file. See “-e mlsrv16 option” on page 55.</td>
</tr>
<tr>
<td>-esu</td>
<td>Use snapshot isolation for uploads. See “-esu mlsrv16 option” on page 56.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>-et filename</code></td>
<td>Store remote database error logs sent into the named file, but delete the contents first if the file exists. See “-et mlsrv16 option” on page 57.</td>
</tr>
<tr>
<td><code>-fips</code></td>
<td>Forces all secure MobiLink streams to be FIPS-certified. See “-fips mlsrv16 option” on page 57.</td>
</tr>
<tr>
<td><code>-ftr path</code></td>
<td>Specifies a location for files that are to be downloaded by the MobiLink transfer utility (mlfiletransfer). See “-ftr mlsrv16 option” on page 58.</td>
</tr>
<tr>
<td><code>-ftru</code></td>
<td>Specifies a location for files uploaded with the MobiLink File Transfer utility (mlfiletransfer). See “-ftru mlsrv16 option” on page 58.</td>
</tr>
<tr>
<td><code>-lsc protocol[protocol-options]</code></td>
<td>Specifies the local server connect information. See “-lsc mlsrv16 option” on page 59.</td>
</tr>
<tr>
<td><code>-nc connections</code></td>
<td>Sets maximum number of concurrent network connections. See “-nc mlsrv16 option” on page 59.</td>
</tr>
<tr>
<td><code>-notifier</code></td>
<td>Starts a Notifier for server-initiated synchronization. See “-notifier mlsrv16 option” on page 60.</td>
</tr>
<tr>
<td><code>-o logfile</code></td>
<td>Log messages to a file. See “-o mlsrv16 option” on page 61.</td>
</tr>
<tr>
<td><code>-on size</code></td>
<td>Set maximum size for log file. See “-on mlsrv16 option” on page 61.</td>
</tr>
<tr>
<td><code>-oq</code></td>
<td>Prevent the popup window on startup error. See “-oq mlsrv16 option” on page 62.</td>
</tr>
<tr>
<td><code>-os size</code></td>
<td>Maximum size of old log files. See “-os mlsrv16 option” on page 62.</td>
</tr>
<tr>
<td><code>-ot logfile</code></td>
<td>Log messages to a file, but delete its contents first. See “-ot mlsrv16 option” on page 63.</td>
</tr>
<tr>
<td><code>-ppv period</code></td>
<td>Causes MobiLink to print new periodic monitoring values according to the period specified. See “-ppv mlsrv16 option” on page 63.</td>
</tr>
<tr>
<td><code>-q</code></td>
<td>Minimize the MobiLink server messages window. See “-q mlsrv16 option” on page 68.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>-r retries</code></td>
<td>Retry deadlocked uploads at most this many times. See “-r mlsrv16 option” on page 68.</td>
</tr>
<tr>
<td><code>-rd delay</code></td>
<td>Set maximum delay, in seconds, before retrying a dead-locked transaction. See “-rd mlsrv16 option” on page 69.</td>
</tr>
<tr>
<td><code>-rp directory</code></td>
<td>Specifies the directory to which synchronizations are recorded for playback with the mlreplay utility. See “-rp mlsrv16 option” on page 70.</td>
</tr>
<tr>
<td><code>-rrp directory</code></td>
<td>Causes the MobiLink server to run the mlreplay utility in the given directory when the server starts. See “-rrp mlsrv16 option” on page 69.</td>
</tr>
<tr>
<td><code>-s count</code></td>
<td>Specify the maximum number of rows to be uploaded to or fetched from the consolidated database at once. See “-s mlsrv16 option” on page 71.</td>
</tr>
<tr>
<td><code>-sl dnet script-options</code></td>
<td>Set the .NET Common Language Runtime (CLR) options and force the CLR to load on startup. See “-sl dnet mlsrv16 option” on page 71.</td>
</tr>
<tr>
<td><code>-sl java script-options</code></td>
<td>Set the Java VM options and force loading of the Java VM on startup. See “-sl java mlsrv16 option” on page 72.</td>
</tr>
<tr>
<td><code>-sm number</code></td>
<td>Set the maximum number of synchronizations that can be worked on concurrently. See “-sm mlsrv16 option” on page 74.</td>
</tr>
<tr>
<td><code>-tc minutes</code></td>
<td>Set the timeout threshold for SQL script execution. See “-tc mlsrv16 option” on page 75.</td>
</tr>
<tr>
<td><code>-tf</code></td>
<td>Fail the SQL script execution when the timeout threshold is reached (not for Oracle). See “-tf mlsrv16 option” on page 76.</td>
</tr>
<tr>
<td><code>-ts session-name(session-option=[option-value,...])</code></td>
<td>Sets up a MobiLink server tracing session. See “-ts mlsrv16 option” on page 76.</td>
</tr>
<tr>
<td><code>-tx count</code></td>
<td>For transactional uploads, batches groups of transactions and commits them together. See “-tx mlsrv16 option” on page 78.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td>-ud</td>
<td>On Unix platforms, run as a daemon. See “-ud mlsrv16 option” on page 78.</td>
</tr>
<tr>
<td>-ui</td>
<td>For Linux with X windows, starts the MobiLink server in shell mode if a usable display isn't available. See “-ui mlsrv16 option” on page 78.</td>
</tr>
<tr>
<td>-ux</td>
<td>For Linux with X windows, opens the MobiLink server messages window. See “-ux mlsrv16 option” on page 79.</td>
</tr>
<tr>
<td>-v [ levels ]</td>
<td>Controls the type of messages written to the log file. See “-v mlsrv16 option” on page 79.</td>
</tr>
<tr>
<td>-w count</td>
<td>Set the initial number of database worker threads. See “-w mlsrv16 option” on page 83.</td>
</tr>
<tr>
<td>-wm count</td>
<td>Set the maximum number of database worker threads. See “-wm mlsrv16 option” on page 84.</td>
</tr>
<tr>
<td>-wn count</td>
<td>Set the number of network worker threads for concurrent processing of network streams. See “-wn mlsrv16 option” on page 75.</td>
</tr>
<tr>
<td>-wu count</td>
<td>Set the maximum number of database worker threads permitted to process uploads concurrently. See “-wu mlsrv16 option” on page 84.</td>
</tr>
<tr>
<td>-x protocol&quot;options; ...&quot;</td>
<td>Specify the communications protocol. Optionally, specify network parameters in form parameter=value, with multiple parameters separated by semicolons. See “-x mlsrv16 option” on page 85.</td>
</tr>
<tr>
<td>-zf</td>
<td>Specifies that the MobiLink server should check for script changes at the beginning of each synchronization. See “-zf mlsrv16 option” on page 94.</td>
</tr>
<tr>
<td>-zp</td>
<td>Ignore some apparent differences between TIMESTAMP values when the remote and consolidated databases have different precision. See “-zp mlsrv16 option” on page 94.</td>
</tr>
<tr>
<td>-zs name</td>
<td>Specify a server name. See “-zs mlsrv16 option” on page 95.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>-zt number</code></td>
<td>Specify the maximum number of processors used to run the MobiLink server. See “<code>-zt mlsrv16 option</code>” on page 95.</td>
</tr>
<tr>
<td>`-zu { +</td>
<td>- }`</td>
</tr>
<tr>
<td><code>-zus</code></td>
<td>Causes MobiLink to invoke upload scripts for tables for which there is no upload. See “<code>-zus mlsrv16 option</code>” on page 96.</td>
</tr>
<tr>
<td><code>-zw 1,...5</code></td>
<td>Controls which levels of warning message to display. See “<code>-zw mlsrv16 option</code>” on page 96.</td>
</tr>
<tr>
<td><code>-zwd code</code></td>
<td>Disables specific warning codes. See “<code>-zwd mlsrv16 option</code>” on page 97.</td>
</tr>
<tr>
<td><code>-zwe code</code></td>
<td>Enables specific warning codes. See “<code>-zwe mlsrv16 option</code>” on page 97.</td>
</tr>
</tbody>
</table>

**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 27

**@data mlsrv16 option**

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

**Syntax**

`mlsrv16 -c "connection-string" @data ...`
Remarks
Use this option to read in mlsrv16 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.

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 also
- “Configuration files” [SQL Anywhere Server - Database Administration]
- “File Hiding utility (dbfhide)” [SQL Anywhere Server - Database Administration]

-a mlsrv16 option
Instructs the MobiLink server to keep using a consolidated database connection after a synchronization error on that connection.

Syntax
mlsrv16 -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 mlsrv16 option
For columns of type VARCHAR, CHAR, LONG VARCHAR, or LONG CHAR, removes trailing blanks from strings during synchronization.

Syntax
mlsrv16 -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 database 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 403
- “upload_fetch_column_conflict table event” on page 405

Example

If the -b option is not used, a primary-key value of 'abc' uploaded from a SQL Anywhere or UltraLite remote database 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 database as 'abc' followed by seven spaces. If the remote database is not blank-padded, then the remote database 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 database 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 database as 'abc'. The -b option fixes the duplicate row problem.

-bn mlsrv16 option

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

Syntax

mlsrv16 -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 mlsrv16 option

Specifies connection parameters for the consolidated database.

Syntax

mlsrv16 -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

mlsrv16 -c "DSN=odbcname;UID=DBA;PWD=sql"

-ca mlsrv16 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

mlsrv16 -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.

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

Note

Port 4953 has been assigned to the MobiLink arbiter so this port number cannot be used by any other applications on the computer where the MobiLink arbiter server is running.
See also
● “-lsc mlsrv16 option” on page 59
● “-notifier mlsrv16 option” on page 60

-cinit mlsrv16 option
Sets the initial size for the server memory cache.

Syntax
mlsrv16 -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 mlsrv16 option” on page 50
● “-cmin mlsrv16 option” on page 50
● “Memory use in the MobiLink server” on page 37

-cm mlsrv16 option
Sets the maximum size for the server memory cache. This option is an alias for the -cmax option. See “-cmax mlsrv16 option” on page 50.
-cmax mlsrv16 option

Sets the maximum size for the server memory cache.

Syntax

mlsrv16 -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 mlsrv16 option” on page 49
- “-cmin mlsrv16 option” on page 50
- “Memory use in the MobiLink server” on page 37

-cmin mlsrv16 option

Sets the minimum size for the server memory cache.

Syntax

mlsrv16 -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 \textit{size} is the amount of memory to reserve in bytes. Use \texttt{k}, \texttt{m}, or \texttt{g} to specify units of kilobytes, megabytes, or gigabytes, respectively. If no letter follows the number, the size is in bytes.

The unit \texttt{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 \texttt{50m}.

\textbf{See also}

- “-cmax mlsrv16 option” on page 50
- “-cinit mlsrv16 option” on page 49
- “Memory use in the MobiLink server” on page 37

\textbf{-cn mlsrv16 option}

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

\textbf{Syntax}

\texttt{mlsrv16 -c "connection-string" -cn value ...}

\textbf{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 mlsrv16 option" on page 83

-cr mlsrv16 option

Sets the maximum number of database connection retries.

Syntax

    mlsrv16 -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 mlsrv16 option

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

Syntax

    mlsrv16 -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, and so on. 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 mlsrv16 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

    mlsrv16 -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 mlsrv16 option
Displays all MobiLink server messages on screen in the MobiLink server messages window.

Syntax

    mlsrv16 -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 mlsrv16 option” on page 61
- “How to log database server messages to a file” [SQL Anywhere Server - Database Administration]

-dr mlsrv16 option
For Adaptive Server Enterprise only. Ensures that tables involved in synchronization do not use the DataRow locking scheme.

Syntax

    mlsrv16 -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 129
-ds mlsrv16 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

```
mlsrv16 -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`.

While holding data for a restartable download, the MobiLink server considers the synchronization to still be active (in the send_download phase of the MobiLink Profiler), and exceeding the network timeout does not close the synchronization.

See also

- “Resumption of failed downloads” on page 124
- “-dc dbmlsync option” [MobiLink - Client Administration]
- “Resume Partial Download synchronization parameter” [UltraLite - Database Management and Reference]

-dsd mlsrv16 option

Disables snapshot isolation.

Syntax

```
mlsrv16 -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. 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 129
- “-dt mlsrv16 option” on page 55
- “-esu mlsrv16 option” on page 56

-dt mlsrv16 option
For Microsoft SQL Server and Adaptive Server Enterprise databases only. Causes MobiLink to detect transactions only within the current database.

Syntax
mlsrv16 -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 129
- “-dsd mlsrv16 option” on page 54
- “-esu mlsrv16 option” on page 56

-e mlsrv16 option
Stores error logs sent from SQL Anywhere MobiLink clients.
Syntax

mlsrv16 -c "connection-string" -e filename ...

Remarks

With no -e option, error logs from SQL Anywhere MobiLink clients are stored in a file named mlsrv16.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 mlsrv16 option” on page 57
- “ErrorLogSendLimit (el) extended option” [MobiLink - Client Administration]

-esu mlsrv16 option

Use snapshot isolation for uploads.

Syntax

mlsrv16 -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 129
- “-dsd mlsrv16 option” on page 54
- “-dt mlsrv16 option” on page 55
- “-r mlsrv16 option” on page 68
- “-rd mlsrv16 option” on page 69
-et mlsrv16 option

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

Syntax

mlsrv16 -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 mlsrv16 option” on page 55

-fips mlsrv16 option

Forces all secure MobiLink streams to use FIPS-certified modules.

Syntax

mlsrv16 -c connection-string" -fips ...

Remarks

Specifying this option forces all MobiLink encryption to use FIPS-certified 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-certified algorithms are used for connections regardless of whether you specify them. 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.

Note

Separately licensed component required.

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

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

For MobiLink transport-layer security, the -fips option causes the server to use the FIPS-certified RSA encryption algorithm, even if RSA without the -fips option is specified.
See also

- “MobiLink client/server communications encryption” [SQL Anywhere Server - Database Administration]
- “FIPS-certified encryption technology” [SQL Anywhere Server - Database Administration]

-ftr mlsrv16 option

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

Syntax

```bash
mlsrv16 -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 to use the mlfiletransfer utility to download files.

See also

- “MobiLink File Transfer utility (mlfiletransfer)” [MobiLink - Client Administration]
- “-ftru mlsrv16 option” on page 58
- “authenticate_file_transfer connection event” on page 260

-ftru mlsrv16 option

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

Syntax

```bash
mlsrv16 -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 to use the mlfiletransfer utility to upload files.
See also

- “MobiLink File Transfer utility (mlfiletransfer)” [MobiLink - Client Administration]
- “-ftr mlsrv16 option” on page 58
- “authenticate_file_transfer connection event” on page 260

-**lsc mlsrv16 option**

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

Syntax

```
mlsrv16 -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 in a MobiLink server farm.
- When using the mlreplay utility with the -rrp server option.

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

```
mlsrv16 -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 connection string tcpip(host=server_rack10;port=200) and use it to connect to the server just started.

See also

- “-rrp mlsrv16 option” on page 69
- “MobiLink server in a server farm” on page 36
- “-zs mlsrv16 option” on page 95
- “-rrp mlsrv16 option” on page 69
- “Notifiers in a MobiLink server farm” [MobiLink - Server-Initiated Synchronization]

-**nc mlsrv16 option**

Sets the maximum number of concurrent network connections.

Syntax

```
mlsrv16 -c "connection-string" -nc connections ...
```

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
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.

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.

See also
- “-sm mlsrv16 option” on page 74

-notifier mlsrv16 option
Starts the Notifier for server-initiated synchronization.

Syntax
mlsrv16 -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]
- “Server-side settings configured 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 mlsrv16 option

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

Syntax

mlsrv16 -c "connection-string" -o logfile ...

Remarks

Write all log messages to the specified file. 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 mlsrv16 option” on page 62
- “-dl mlsrv16 option” on page 53
- “-ot mlsrv16 option” on page 63
- “-on mlsrv16 option” on page 61
- “-v mlsrv16 option” on page 79

-on mlsrv16 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

```
mlsrv16 -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 mlsrv16 option” on page 61
- “-ot mlsrv16 option” on page 63
- “-on mlsrv16 option” on page 61
- “-os mlsrv16 option” on page 62
- “-v mlsrv16 option” on page 79

-**-oq** mlsrv16 option

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

Syntax

```
mlsrv16 -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** mlsrv16 option

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

Syntax

```
mlsrv16 -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 `ymmdxx.mls`, where `xx` is a number from 00 to 99, and `ymmd` 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 mlsrv16 option” on page 61
- “-on mlsrv16 option” on page 61
- “-ot mlsrv16 option” on page 63
- “-v mlsrv16 option” on page 79

**-ot mlsrv16 option**

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

**Syntax**

```
mlsrv16 -c "connection-string" -ot logfilename ...
```

**Remarks**

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

**See also**

- “-on mlsrv16 option” on page 61
- “-os mlsrv16 option” on page 62
- “-v mlsrv16 option” on page 79
- “-o mlsrv16 option” on page 61

**-ppv mlsrv16 option**

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

**Syntax**

```
mlsrv16 -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:

<table>
<thead>
<tr>
<th>Printed value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMD_PROCESSOR_STAGE_LEN</td>
<td>The length of the queue for synchronization work.</td>
</tr>
<tr>
<td>CPU_USAGE</td>
<td>The amount of CPU time used by the MobiLink server in microseconds.</td>
</tr>
<tr>
<td>DB_CONNECTIONS</td>
<td>The number of database connections in use.</td>
</tr>
<tr>
<td>FREE_DISK_SPACE</td>
<td>The disk space available on the temp disk in bytes.</td>
</tr>
<tr>
<td>HEARTBEAT_STAGE_LEN</td>
<td>The length of the queue for periodic, non-sync work.</td>
</tr>
<tr>
<td>LONGEST_DB_WAIT</td>
<td>The longest length of time an active synchronization has been waiting for the database.</td>
</tr>
<tr>
<td>LONGEST_SYNC</td>
<td>The age of the oldest synchronization in microseconds.</td>
</tr>
<tr>
<td>MEMORY_USED</td>
<td>The bytes of RAM in use (for Windows only).</td>
</tr>
<tr>
<td>ML_NUM_CONNECTED_CLIENTS</td>
<td>The number of connected synchronization clients.</td>
</tr>
<tr>
<td>NOTIFIER_STAGE_LEN</td>
<td>The length of the notifier work queue.</td>
</tr>
<tr>
<td>NUM_COMMITS</td>
<td>The total number of commits.</td>
</tr>
<tr>
<td>NUM_CONNECTED_FILE_XFERS</td>
<td>The number of mlfiletransfers currently connected.</td>
</tr>
<tr>
<td>NUM_CONNECTED_LISTENERS</td>
<td>The number of listeners currently connected.</td>
</tr>
<tr>
<td>NUM_CONNECTED_MONITORS</td>
<td>The number of monitors currently connected.</td>
</tr>
<tr>
<td>NUM_CONNECTED_PINGS</td>
<td>The number of pinging clients currently connected.</td>
</tr>
<tr>
<td>NUM_CONNECTED_SYNCS</td>
<td>The number of data synchronizations currently connected.</td>
</tr>
<tr>
<td>Printed value</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NUM_ERRORS</td>
<td>The total number of errors.</td>
</tr>
<tr>
<td>NUM_FAILED_SYNCS</td>
<td>The total number of failed syncs.</td>
</tr>
<tr>
<td>NUM_IN_APPLY_UPLOAD</td>
<td>The number of synchronizations currently in the apply upload phase.</td>
</tr>
<tr>
<td>NUM_IN_AUTH_USER</td>
<td>The number of synchronizations currently in the authenticate user phase.</td>
</tr>
<tr>
<td>NUM_IN_BEGIN_SYNC</td>
<td>The number of synchronizations currently in the begin synchronization phase.</td>
</tr>
<tr>
<td>NUM_IN_CONNECT</td>
<td>The number of synchronizations currently in the connect phase.</td>
</tr>
<tr>
<td>NUM_IN_CONNECT_FOR_ACK</td>
<td>The number of synchronizations currently in the connect for download ack phase.</td>
</tr>
<tr>
<td>NUM_IN_END_SYNC</td>
<td>The number of synchronizations currently in the end synchronization phase.</td>
</tr>
<tr>
<td>NUM_IN_FETCH_DNLD</td>
<td>The number of synchronizations currently in the fetch download phase.</td>
</tr>
<tr>
<td>NUM_IN_GET_DB_WORKER_FOR_ACK</td>
<td>The number of synchronizations currently waiting for a database connection to process a non-blocking download acknowledgement.</td>
</tr>
<tr>
<td>NUM_IN_NON_BLOCKING_ACK</td>
<td>The number of synchronizations currently in the non-blocking download ack phase.</td>
</tr>
<tr>
<td>NUM_IN_PREP_FOR_DNLD</td>
<td>The number of synchronizations currently in the prepare for download phase.</td>
</tr>
<tr>
<td>NUM_IN_RECVING_UPLOAD</td>
<td>The number of synchronizations currently in the receive upload phase.</td>
</tr>
<tr>
<td>NUM_IN_SEND_DNLD</td>
<td>The number of synchronizations currently in the send download phase.</td>
</tr>
<tr>
<td>NUM_IN_SYNC_REQUEST</td>
<td>The number of synchronizations currently in the synchronization request phase.</td>
</tr>
<tr>
<td>NUM_IN_WAIT_FOR_DNLD_ACK</td>
<td>The number of synchronizations currently in the wait for download ack phase.</td>
</tr>
<tr>
<td><strong>Printed value</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NUM_ROLBACKS</td>
<td>The total number of rollbacks.</td>
</tr>
<tr>
<td>NUM_ROWS_DOWNLOADED</td>
<td>The total number of rows sent to remotes.</td>
</tr>
<tr>
<td>NUM_ROWS_UPLOADED</td>
<td>The total number of rows received from remotes.</td>
</tr>
<tr>
<td>NUM_SUCCESS_SYNCS</td>
<td>The total number of successful syncs.</td>
</tr>
<tr>
<td>NUM_UPLOAD_CONNS_IN_USE</td>
<td>The number of upload connections currently in use.</td>
</tr>
<tr>
<td>NUM_WAITING_CONS</td>
<td>The number of synchronizations currently waiting for the consolidated database.</td>
</tr>
<tr>
<td>NUM_WARNINGS</td>
<td>The total number of warnings.</td>
</tr>
<tr>
<td>OE_STAGE_LEN</td>
<td>The length of the integrated Outbound Enabler work queue.</td>
</tr>
<tr>
<td>PAGES_IN_STREAMSTACK</td>
<td>The number of pages held by the network streams.</td>
</tr>
<tr>
<td>PAGES_LOCKED</td>
<td>The number of cache pages loaded into memory.</td>
</tr>
<tr>
<td>PAGES_LOCKED_MAX</td>
<td>The number of pages in the memory cache.</td>
</tr>
<tr>
<td>PAGES_SWAPPED_IN</td>
<td>The total number of pages ever read from disk.</td>
</tr>
<tr>
<td>PAGES_SWAPPED_OUT</td>
<td>The total number of pages ever swapped to disk.</td>
</tr>
<tr>
<td>PAGES_USED</td>
<td>The number of cache pages used. This includes pages swapped to disk so it may be larger than the cache size.</td>
</tr>
<tr>
<td>PRIMARY_IS_KNOWN</td>
<td>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 is. Shows 2 if the server does not know what the primary server is.</td>
</tr>
<tr>
<td>RAW_TCP_STAGE_LEN</td>
<td>The length of the network work queue.</td>
</tr>
<tr>
<td>SERVER_IS_PRIMARY</td>
<td>Indicates if the server is primary or secondary. Shows 1 if the server is primary, otherwise shows 0.</td>
</tr>
<tr>
<td>SIRT_NUM_LWP_HITS</td>
<td>The number of lightweight polls from remote task agents, indicating a notification.</td>
</tr>
<tr>
<td>SIRT_NUM_LWPS</td>
<td>The number of lightweight polls from remote task agents.</td>
</tr>
<tr>
<td>Printed value</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SIRT_NUM_REQUESTS</td>
<td>The number of remote task notifications currently outstanding.</td>
</tr>
<tr>
<td>STREAM_STAGE_LEN</td>
<td>The length of the high level network processing queue.</td>
</tr>
<tr>
<td>TCP_BYTES_READ</td>
<td>The total number of bytes ever read.</td>
</tr>
<tr>
<td>TCP_BYTES_WRITTEN</td>
<td>The total number of bytes ever written.</td>
</tr>
<tr>
<td>TCP_CONNECTIONS</td>
<td>The number of TCP connections currently opened.</td>
</tr>
<tr>
<td>TCP_CONNECTIONS_CLOSED</td>
<td>The total number of connections ever closed.</td>
</tr>
<tr>
<td>TCP_CONNECTIONS_OPENED</td>
<td>The total number of connections ever opened.</td>
</tr>
<tr>
<td>TCP_CONNECTIONS_REJECTED</td>
<td>The total number of connections ever rejected.</td>
</tr>
<tr>
<td>TIMED_WORK_STAGE_LEN</td>
<td>The length of the dynamic caching work queue.</td>
</tr>
<tr>
<td>TRACKED_MEMORY</td>
<td>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.</td>
</tr>
<tr>
<td>VM_MEM_USE</td>
<td>The amount of memory used by any attached VMs.</td>
</tr>
</tbody>
</table>

Example

Below is sample output showing the periodic monitoring values.

```
I. 2009-10-28 11:46:29. <Main> PERIODIC: PAGES_IN_STREAMSTACK: 0
```
-q mlsrv16 option
Instructs MobiLink to run with a minimized messages window on startup.

Syntax
mlsrv16 -c "connection-string" -q ...

Remarks
Minimize the MobiLink server messages window.

-r mlsrv16 option
Sets the maximum number of deadlock retries.
Syntax
mlsrv16 -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 mlsrv16 option
Sets the maximum delay time between deadlock retries.

Syntax
mlsrv16 -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 mlsrv16 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
mlsrv16 -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.

To use the -rrp and -rp options:

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

See also

- “-rp mlsrv16 option” on page 70
- “-lsc mlsrv16 option” on page 59
- “MobiLink Replay utility (mlreplay)” on page 656

-rp mlsrv16 option

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

Syntax

mlsrv16 -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.

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

See also

- “-rrp mlsrv16 option” on page 69
- “MobiLink Replay utility (mlreplay)” on page 656
-s mlsrv16 option

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

Syntax

```
mlsrv16 -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 mlsrv16 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

```
mlsrv16 -c "connection-string" -sl dnet options ...
```

Remarks

Sets options to pass directly to the .NET CLR.

The available options are as follows:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-D name=value</code></td>
<td>Set an environment variable. For example: <code>-D synchtype=far</code> <code>-D extra_rows=yes</code></td>
</tr>
<tr>
<td><code>-MLAutoLoadPath=path</code></td>
<td>Set the location of base assemblies. Only works with private assemblies. To tell MobiLink where assemblies are located, use this option or <code>-MLDomConfigFile</code>, but not both. When you use <code>-MLAutoLoadPath</code>, you cannot specify a domain in the event script. The default is the current directory.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-MLDomConfigFile= file</td>
<td>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. When the file path referenced in the -MLDomConfigFile option refers to a file in a folder with a space in the name, such as &quot;C:\Program Files\MyCompany\SyncServer\MLDomConfig.xml&quot;, place double quotes around the whole option: &quot;-MLDomConfigFile=C:\Program Files\MyCompany\SyncServer\MLDomConfig.xml&quot;.</td>
</tr>
<tr>
<td>-MLStartClasses= classnames</td>
<td>At server startup, load and instantiate user-defined start classes in the order listed.</td>
</tr>
<tr>
<td>-clrConGC</td>
<td>Enable concurrent garbage collection in the CLR.</td>
</tr>
<tr>
<td>-clrFlavor=( wks</td>
<td>svr )</td>
</tr>
<tr>
<td>-clrVersion= version</td>
<td>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. For example, v4.0.30319. To use v4.0 assemblies, you need to explicitly add the -clrVersion option to make the MobiLink server load a v4.0 runtime. For example, -clrVersion=v4.0.30319. To display this list of options, run the following command: mlsrv16 -sl dnet (?) See also ● “Synchronization scripts in .NET” on page 497</td>
</tr>
</tbody>
</table>

**-sl java mlsrv16 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**

mlsrv16 -c "connection-string" -sl java (options) ...
Remarks
Sets -jrepath and other options to pass directly to the Java VM. The options are:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-classic</td>
<td>Use the classic Java VM.</td>
</tr>
<tr>
<td>-client</td>
<td>Use the client Java VM.</td>
</tr>
<tr>
<td>-hotspot</td>
<td>Use the hotspot Java VM.</td>
</tr>
<tr>
<td>-server</td>
<td>Use the server Java VM. This is the default.</td>
</tr>
<tr>
<td>-cp location;...</td>
<td>Specify a set of directories or JAR files in which to search for classes.</td>
</tr>
<tr>
<td>-D name=value</td>
<td>Set a system property. For example:</td>
</tr>
<tr>
<td>-DMLStartClasses=classname, ...</td>
<td>At server startup, load and instantiate user-defined start classes in the order listed.</td>
</tr>
<tr>
<td>-jrepath path</td>
<td>Override the default JRE path, which is one of the following directories:</td>
</tr>
<tr>
<td></td>
<td>%SQLANY16%\Bin32\jre170 or %SQLANY16%\Bin64\jre170, for 32-bit and 64-bit platforms, respectively.</td>
</tr>
<tr>
<td>-X vm-option</td>
<td>Set a VM-specific option as described in the file %SQLANY16%\Bin32\jre170\bin\client\Xusage.txt or %SQLANY16%\Bin64\jre170\bin\server\Xusage.txt, for 32-bit and 64-bit platforms, respectively.</td>
</tr>
</tbody>
</table>

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

```bash
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, 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
- “Synchronization script writing in Java” on page 427
Examples

For example, on Windows the following partial mlsrv16 command line sets the Java VM option that enables system asserts:

```
mlsrv16 -sl java (-cp ;\myclasses; -esa) ...
```

On Windows, the following partial mlsrv16 command line defines the LDAP_SERVER system property:

```
mlsrv16 -sl java ( -cp ;\myclasses; -DLDAP_SERVER=mycorp-ldap ) ...
```

The following partial mlsrv16 command line works on Unix:

```
mlsrv16 -sl java { -cp .:$CLASSPATH:/opt/myclasses:/opt/my.jar: }
```

-sm mlsrv16 option

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

Syntax

```
mlsrv16 -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 mlsrv16 -w option.
- The number of synchronizations doing task 2 is less than or equal to the setting for the mlsrv16 -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 mlsrv16 option” on page 83
- “-wu mlsrv16 option” on page 84
- “-nc mlsrv16 option” on page 59

-wn mlsrv16 option

Sets the number of network worker threads the MobiLink server uses for concurrent processing of network streams.

Syntax

```
mlsrv16 -c "connection-string" -wn count ...
```

Remarks

The default value is 1.

Having multiple network worker threads can improve performance, particularly when using CPU-intensive network stream options, like encryption or compression, with either large synchronizations or many small synchronizations. Each request in the system can be active on one network stream thread, at most.

-tc mlsrv16 option

Sets a timeout threshold for long running SQL scripts.

Syntax

```
mlsrv16 -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 mlsrv16 option” on page 76

-tf mlsrv16 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

```
mlsrv16 -c "connection string" -tf ...
```

Remarks

If the SQL script fails, the MobiLink server either skips the row (if the script is an upload script and if the handle_error script returns 1000) and continues the synchronization, or aborts 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.

-ts mlsrv16 option

Sets up a MobiLink server tracing session.

Syntax

```
mlsrv16 -c "connection-string" -ts session-name(session-option=[option-value;...])
```

The session name must be logging.

<table>
<thead>
<tr>
<th>Session option</th>
<th>Option value</th>
</tr>
</thead>
<tbody>
<tr>
<td>events</td>
<td>Comma separated list of system trace events. The supported events are Info, Warning, and Error.</td>
</tr>
</tbody>
</table>
The target options are specified as name-value pairs. The target file can have the following options:

<table>
<thead>
<tr>
<th>Target option name</th>
<th>Expected value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>filename_prefix</td>
<td>String</td>
<td>An ETD file name prefix with or without a path. All ETD files have the extension .etd. This parameter is required.</td>
</tr>
<tr>
<td>max_size</td>
<td>Integer</td>
<td>The maximum size of the file in bytes. The default is 0, which means there is no limit on the file size and it grows as long as disk space is available. Once the specified size is reached, a new file is started.</td>
</tr>
<tr>
<td>num_files</td>
<td>Integer</td>
<td>The number of files where event tracing information is written, and it is used only if max_size is set. If all the files reach the maximum specified size, the MobiLink server starts overwriting the oldest file.</td>
</tr>
<tr>
<td>flush_on_write</td>
<td>yes, true, no, false</td>
<td>A value that controls whether disk buffers are flushed for each event that is logged. The values yes, true, no, and false are accepted. The default is false. When this parameter is turned on, the performance of the MobiLink server may be reduced if many trace events are being logged.</td>
</tr>
<tr>
<td>compressed</td>
<td>yes, true, no, false</td>
<td>A value that controls compression of the ETD file to conserve disk space. The default is false.</td>
</tr>
</tbody>
</table>

Remarks

All information specified after the `-ts logging` portion of the option must be specified without any spaces.

See also

- “Event Trace Data (ETD) File Management utility (dbmanageetd)” [SQL Anywhere Server - Database Administration]

Example

Following is an example of the -ts option:

```
-ts logging{events=Info,warning,Error;targets=file(filename_prefix=mls_etd;max_size=10000000;num_files=10;flush_on_write=true)}
```
-tx mlsrv16 option

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

Syntax
mlsrv16 -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 mlsrv16 option

Instructs MobiLink to run as a daemon.

Syntax
mlsrv16 -c "connection-string" -ud ...

Remarks
This option applies to Unix platforms only.

See also
- “MobiLink server use outside the current session” on page 31

-ui mlsrv16 option

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

Syntax
mlsrv16 -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 mlsrv16 option

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

Syntax

\texttt{mlsrv16 \texttt{-c} "connection-string \texttt{-ux} \ldots"}

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

- \textit{“-q mlsrv16 option” on page 68}

-v mlsrv16 option

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

Syntax

\texttt{mlsrv16 \texttt{-c} "connection-string \texttt{-v[ levels]} \ldots"}

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 \texttt{ml_property} table every five minutes and looks for verbose settings for a MobiLink user or remote ID. See \textit{“Log verbosity for targeted MobiLink users and remote IDs” on page 623}.

When a \texttt{CHAR}, \texttt{VARCHAR}, \texttt{NCHAR} or \texttt{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 lowercase verbosity levels.

- Show the content of each synchronization script when it is invoked. This level implies `s`.

- Show system event scripts. These system event scripts are used to query and maintain MobiLink system tables.

- 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 mlsrv16 option” on page 85.

- Show the remote schema being synchronized.

- 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`.

- 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.

- 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 Profiler. 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. The fetch download phase does not include the time to create the download stream, which is done in the send download phase. This can take a significant amount of time for large downloads, when the download cannot fit in memory.

○ **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.

The send download phase includes the time to create the download stream, which can take a significant amount of time for large downloads, when the download cannot fit in memory.

○ **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: sync_request: 0
I. 2008-06-05 14:48:36. <1> PHASE: get_db_worker: 0
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: end_sync: 0
I. 2008-06-05 14:48:36. <1> PHASE: send_download: 10
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.

```sql
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.

```sql
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 synchronization statistical properties” on page 191

-w mlsrv16 option

Sets the initial number of concurrent database worker threads, up to the number of threads specified with the -wm option.

Syntax

mlsrv16 -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.

The default number of database worker threads is 5.

See also

- “-wm mlsrv16 option” on page 84
- “-wu mlsrv16 option” on page 84
- “-sm mlsrv16 option” on page 74
- “-cn mlsrv16 option” on page 51
**-wm mlsrv16 option**

Sets the maximum number of concurrent database worker threads.

**Syntax**

```
mlsrv16 -c "connection-string" -wm count ...
```

**Remarks**

The MobiLink server monitors performance and automatically adjusts the number of database worker threads as necessary. The MobiLink server uses any value between the initial value, set with the -w option, and the maximum value, set with the -wm option.

This feature allows deployments to achieve better throughput with less load testing. Given a wide enough range between -w and -wm, MobiLink server automatically finds the number of database worker threads providing the best throughput. However, the heuristic used to adjust the number of database worker threads may not work well in all cases. Also, the best throughput may lie outside the limits set by -w and -wm. Only deployment-specific load testing can truly establish the number of database worker threads to provide maximum throughput.

If this value is not set, the maximum number of database worker threads default to the value set in the -w option. When the -wm option is not used, the number of database worker threads is fixed at the -w value, and the MobiLink server does not automatically adjust them.

**See also**

- “-w mlsrv16 option” on page 83
- “-wu mlsrv16 option” on page 84
- “-sm mlsrv16 option” on page 74
- “-cn mlsrv16 option” on page 51
- “Automatic adjustment of database worker threads” on page 133

**-wu mlsrv16 option**

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

**Syntax**

```
mlsrv16 -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 mlsrv16 -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.

If -wu is not specified, uploads may be applied concurrently on any or all database worker threads. If -wu is specified, uploads are only applied concurrently on the specified number of database worker threads. This may temporarily increase contention as the MobiLink server increases the number of database worker threads in an attempt to increase throughput. When this condition is detected, the thread count is decreased.

Setting -wu is recommended in high-load environments where uploads are constantly intermixed with download-only synchronizations.

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 mlsrv16 option” on page 83
- “-wm mlsrv16 option” on page 84
- “-sm mlsrv16 option” on page 74

-x mlsrv16 option

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

Note

Separately licensed component required.

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

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

Syntax

\[mlsrv16 -c "connection-string" -x protocol[protocol-options] [ -x protocol[protocol-options] ... ] ...\]

\[protocol : tcpip | tls | http | https | oe\]

\[protocol-options : ( option=value; ... )\]

Remarks

The -x option must be specified for each protocol being used. For example, to have MobiLink listen for both TCP/IP and HTTP, you would specify something like the following:
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 encryption.

- **oe**  
  Use an integrated Outbound Enabler when using the Relay Server. You cannot include more than one -x oe protocol 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):

<table>
<thead>
<tr>
<th>TCP/IP protocol option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>collect_network_data</code></td>
<td>Enables synchronization scripts to read network information from each synchronization. See “NetworkData interface [MobiLink server Java]” on page 463 and “NetworkData interface [MobiLink server .NET]” on page 555.</td>
</tr>
<tr>
<td><code>host</code></td>
<td>The host name or IP number on which the MobiLink server should listen. The default value is localhost.</td>
</tr>
<tr>
<td><code>ignore</code></td>
<td>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 Profiler output files. You can specify multiple hosts to ignore; for example <code>-x tcpip(ignore=lb1;ignore=123.45.67.89)</code>. If you specify multiple instances of -x on a command line, the host is ignored on all instances; for example, if you specify <code>-x tcpip(ignore=1.1.1.1) -x http</code>, then connections for 1.1.1.1 are ignored on both the TCP/IP and the HTTP streams.</td>
</tr>
<tr>
<td><code>port</code></td>
<td>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.</td>
</tr>
</tbody>
</table>
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):

<table>
<thead>
<tr>
<th>TLS protocol options</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>collect_network_data={yes</td>
<td>no}</td>
</tr>
<tr>
<td>e2ee_private_key=file</td>
<td>The PEM or DER encoded file containing the RSA private key. This option is required for end-to-end encryption to take effect. PEM and DER encoded files are created using the createkey utility. See “Key Pair Generator utility (createkey)” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>e2ee_private_key_password=password</td>
<td>The password to the private key file. This option is required for end-to-end encryption to take effect. When this option is specified, the e2ee_private_key parameter must also be specified. 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].</td>
</tr>
<tr>
<td>fips={yes</td>
<td>no}</td>
</tr>
<tr>
<td>host=hostname</td>
<td>The host name or IP number on which the MobiLink server should listen. The default value is localhost.</td>
</tr>
<tr>
<td>identity=identity-file</td>
<td>The path and file name of the identity file that is to be used for server authentication.</td>
</tr>
</tbody>
</table>
### TLS protocol options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>identity_password=password</td>
<td>An optional parameter that specifies a password for the identity file. When this option is specified, the identity option must also be specified. 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].</td>
</tr>
<tr>
<td>ignore=hostname</td>
<td>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 Profiler output files. You can specify multiple hosts to ignore; for example -x tcpip(ignore=lb1;ignore=123.45.67.89).</td>
</tr>
<tr>
<td>port=portnumber</td>
<td>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.</td>
</tr>
<tr>
<td>trusted_certificates=certificate_file</td>
<td>Use this option to ensure the client certificate is valid, then use the NetworkData.ClientCertificates API to further authenticate the certificate in the authenticate_user script. See “NetworkData interface [MobiLink server Java]” on page 463 and “NetworkData interface [MobiLink server .NET]” on page 555.</td>
</tr>
</tbody>
</table>

### HTTP options

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer_size=number</td>
<td>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.</td>
</tr>
<tr>
<td>collect_network_data={yes</td>
<td>Enables synchronization scripts to read network information from each synchronization. See “NetworkData interface [MobiLink server Java]” on page 463 and “NetworkData interface [MobiLink server .NET]” on page 555.</td>
</tr>
</tbody>
</table>
### HTTP options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>header_limit=number</td>
<td>The maximum amount of header data that can be sent in an HTTP request. If a request exceeds the value specified, the server returns an HTTP error code and aborts the request. For example, <code>-x http(header_limit=200000)</code> raises the limit to 200000 bytes. The default value is 64000 bytes.</td>
</tr>
<tr>
<td>host=hostname</td>
<td>The host name or IP number on which the MobiLink server should listen. The default value is localhost.</td>
</tr>
<tr>
<td>log_bad_request={yes</td>
<td>no}</td>
</tr>
<tr>
<td>port=portnumber</td>
<td>The socket port number on which the MobiLink server should listen. The default port is 80.</td>
</tr>
<tr>
<td>version=http-version</td>
<td>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.</td>
</tr>
</tbody>
</table>

### HTTPS options

The HTTPS protocol uses RSA 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):

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer_size=number</td>
<td>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.</td>
</tr>
<tr>
<td>collect_network_data={yes</td>
<td>no}</td>
</tr>
<tr>
<td>HTTPS options</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-------------</td>
</tr>
<tr>
<td>e2ee_private_key=file</td>
<td>The PEM or DER encoded file containing the RSA private key. This option is required for end-to-end encryption to take effect. PEM and DER encoded files are created using the createkey utility. See “Key Pair Generator utility (createkey)” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>e2ee_private_key_password=password</td>
<td>The password to the private key file. This option is required for end-to-end encryption to take effect. When this option is specified, the e2ee_private_key option must also be specified. 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].</td>
</tr>
<tr>
<td>fips={yes</td>
<td>no}</td>
</tr>
<tr>
<td>header_limit=number</td>
<td>The maximum amount of header data that can be sent in an HTTPS request. If a request exceeds the value specified, the server returns an error code and aborts the request. For example, <code>-x https(header_limit=200000)</code> raises the limit to 200000 bytes. The default value is 64000 bytes.</td>
</tr>
<tr>
<td>host=hostname</td>
<td>The host name or IP number on which the MobiLink server should listen. The default value is localhost.</td>
</tr>
<tr>
<td>identity=server-identity</td>
<td>The path and file name of the identity file that is to be used for server authentication.</td>
</tr>
<tr>
<td>identity_password=password</td>
<td>An optional parameter that specifies a password for the identity file. When this option is specified, the identity option must also be specified. 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].</td>
</tr>
<tr>
<td>HTTPS options</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>log_bad_request= { yes</td>
<td>no }</td>
</tr>
<tr>
<td></td>
<td>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 mlsrv16 option” on page 79.</td>
</tr>
<tr>
<td>port=portnumber</td>
<td>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.</td>
</tr>
<tr>
<td>trusted_certificates=certificate_file</td>
<td>Use this option to ensure the client certificate is valid, then use the NetworkData.ClientCertificates API to further authenticate the certificate in the authenticate_user script. See “NetworkData interface [MobiLink server Java]” on page 463 and “NetworkData interface [MobiLink server .NET]” on page 555.</td>
</tr>
<tr>
<td>version=http-version</td>
<td>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.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>OE protocol option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>buffer_size=number</td>
<td>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.</td>
</tr>
</tbody>
</table>
| 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 option within the file.  

You must specify exactly one configuration file. |
| header_limit=number     | The maximum amount of header data that can be sent in an OE request. If a request exceeds the value specified, the server returns an error code and aborts the request. For example, -x oe(header_limit=200000) raises the limit to 200000 bytes. The default value is 64000 bytes. |
| 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 mlsrv16 option” on page 79. |
**OE protocol option** | **Description**
--- | ---
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.

### Example

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

```
mlsrv16 -c "DSN=SQL Anywhere 16 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:

```
mlsrv16 -c "DSN=my_cons"
-x tls(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:

```
mlsrv16 -c "DSN=my_cons"
-x "tls(identity=c:\Program Files\test\serv_rsa1.crt;identity_password=pwd)"
```

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

```
mlsrv16 -c "DSN=my_cons" -x https(identity=my_identity.crt;identity_password=my_id_pwd;e2ee_private_key=my_pk.pem;e2ee_private_key_password=my_pk_pwd)
```

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

```
mlsrv16 -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 a18b2e37dbc296322ec5a6eced6ba896
-cr "host=relayserver.sybase.com;https=0;port=80"
```

### trusted_certificates example for Java

The following example shows how to use the `NetworkData` interface to retrieve certificate information from a secure synchronization.

```java
public class OrderProcessor {
    DBConnectionContext _cc;

    public OrderProcessor( DBConnectionContext cc ) {
        _cc = cc;
    }

    // The method used for the authenticate_user event.
    public void AuthUser() {
        NetworkData nd = _cc.getNetworkData();
```
if( nd != null ) {
    if( nd.isTLS() ) {
        CertPath certs = nd.getCertificateChain();
        if( certs != null ) {
            System.out.println( "  client-side cert:" );
            int n = 1;
            for( Certificate c : certs.getCertificates() ) {
                System.out.println( "    cert "+ n++);
                X509Certificate c509 = (X509Certificate) c;
                System.out.println( "      Subject: " +
                                    c509.getSubjectX500Principal().getName() );
                System.out.println( "      Issuer: " +
                                    c509.getIssuerX500Principal().getName() );
            }
        } else {
            System.out.println( "  no client cert" );
        }
    }
}

Execute the following SQL statement to register the Java method.

```
ml_add_java_connection_script( <version>, 'authenticate_user',
    'OrderProcessor.AuthUser' )
```

The following two examples show the options to add to the MobiLink command line. The first example is for HTTPS and the second example is for TLS.

```
mlsrv16 -c <connection_string> -x
https(collect_network_data=1;trusted_certificates=<certificate_file>) -sl java

mlsrv16 -c <connection_string> -x
tls(collect_network_data=1;trusted_certificates=<certificate_file>) -sl java
```

**trusted_certificates example for .NET**

The following example shows how to use the NetworkData interface to retrieve certificate information from a secure synchronization.

```
public class OrderProcessor {
    DBCConnectionContext _cc;

    public OrderProcessor( DBCConnectionContext cc ) {
        _cc = cc;
    }

    public void AuthUser() {
        NetworkData nd = _cc.NetworkData;
        if( nd != null ) {
            if( nd.IsTLS ) {
                X509Certificate2Collection certs = nd.ClientCertificates;
                if( certs != null ) {
                    PrintLn( " client-side cert:" );
                    int n = 1;
                    foreach( X509Certificate2 x509 in certs ) {
                        PrintLn( "    cert "+ n++);
                        PrintLn( "      Subject: " + x509.SubjectName.Name );
                        PrintLn( "      Issuer: " + x509.IssuerName.Name );
                    }
                }
            }
        }
    }
```
Execute the following SQL statement to register the .NET method.

```sql
ml_add_dnet_connection_script( <version>, 'authenticate_user', 'OrderProcessor.AuthUser' )
```

The following two examples show the options to add to the MobiLink command line. The first example is for HTTPS and the second example is for TLS.

```bash
mlsrv16 -c <connection_string> -x https(collect_network_data=1;trusted_certificates=<certificate_file>) -sl dnet
mlsrv16 -c <connection_string> -x tls(collect_network_data=1;trusted_certificates=<certificate_file>) -sl dnet
```

### -zf mlsrv16 option

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

**Caution**
Running the MobiLink server with the -zf option has a negative impact on MobiLink server performance and should be avoided whenever possible.

**Syntax**

```
mlsrv16 -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 mlsrv16 option

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

**Syntax**

```
mlsrv16 -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 database 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 mlsrv16 option

Specifies a MobiLink server name for mlstop.

Syntax

mlsrv16 -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 652

-zt mlsrv16 option

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

Syntax

mlsrv16 -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 mlsrv16 option

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

Syntax

mlsrv16 -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 653
- “authenticate_user connection event” on page 266

-zus mlsrv16 option
Causes the MobiLink server to invoke upload scripts for a table even when no rows are uploaded for the table.

Syntax
mlsrv16 -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 mlsrv16 option
Controls which levels of warning message to display.

Syntax
mlsrv16 -c "connection-string" -zw levels

Remarks
MobiLink has five levels of warning messages:

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Suppress all warning messages</td>
</tr>
<tr>
<td>1</td>
<td>Server and high ODBC level: warning messages when the MobiLink server starts</td>
</tr>
</tbody>
</table>
## Level and Description

<table>
<thead>
<tr>
<th>Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Synchronization and user level: warning messages when a synchronization starts</td>
</tr>
<tr>
<td>3</td>
<td>Schema level: warning messages when a MobiLink server is processing a client schema</td>
</tr>
<tr>
<td>4</td>
<td>Script and lower ODBC level: warning messages when a MobiLink server fetches, prepares, or executes scripts</td>
</tr>
<tr>
<td>5</td>
<td>Table or row level: warning messages when a MobiLink server performs table operations in an upload or download</td>
</tr>
</tbody>
</table>

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 mlsrv16 option

Disables specific warning codes.

**Syntax**

```
mlsrv16 -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 mlsrv16 option

Enables specific warning codes.
Syntax

\texttt{mlsrv16 -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 also

- “Unique primary keys” on page 108
Implementing 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.

Prerequisites

There are no prerequisites for this task.

Context and remarks

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

Task

1. To implement timestamp-based synchronization for a table, 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:

<table>
<thead>
<tr>
<th>DBMS</th>
<th>last modified column</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptive Server Enterprise</td>
<td>datetime</td>
</tr>
<tr>
<td>IBM DB2 LUW</td>
<td>timestamp NOT NULL GENERATED ALWAYS FOR EACH ROW ON UPDATE AS ROW CHANGE TIME-</td>
</tr>
<tr>
<td></td>
<td>STAMP</td>
</tr>
<tr>
<td>Microsoft SQL Server</td>
<td>datetime</td>
</tr>
<tr>
<td>MySQL</td>
<td>timestamp default CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP</td>
</tr>
<tr>
<td>Oracle</td>
<td>timestamp</td>
</tr>
<tr>
<td>SQL Anywhere</td>
<td>timestamp DEFAULT timestamp</td>
</tr>
</tbody>
</table>

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

Results

The timestamp-based synchronization is implemented.

Example

The following example, taken from the MobiLink Contact sample, is an illustration of how you can implement a timestamp-based download.
Table definition:

```sql
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:

```sql
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 also

- “Synchronization of contacts in the Contact sample” [MobiLink - Getting Started]
- “Last download times in scripts” on page 100
- “Synchronization logic source code” [MobiLink - Getting Started]

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.

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

- “How download timestamps are generated and used” on page 101
- “Script parameters” on page 216

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 database 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”
- “generate_next_last_download_timestamp event”
- “modify_next_last_download_timestamp connection event”

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

- “ULResetLastDownloadTime method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- “ULConnection.ResetLastDownloadTime method [UltraLite C++]” [UltraLite - C and C++ Programming]
- “ULConnection.ResetLastDownloadTime method [UltraLite.NET]” [UltraLite - .NET Programming]

See also

- “Last download times in scripts” on page 100

Daylight savings time solutions

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 with a partitioning of the rows. See “Partitioned rows among remote databases” on page 104.

### 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.

### Implementation of snapshot-based synchronization

Keep the following in mind when implementing snapshot-based synchronization.

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

- 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 “download_delete_cursor scripts” on page 244.

- 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 pre-existing row with the same primary key.

See “Scripts to download rows” on page 242.

### 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 “download_delete_cursor scripts” on page 244.

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.

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

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

Partitioned rows among remote databases

Each MobiLink remote database can contain a different subset of the data in the consolidated database. 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 use 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, follow these guidelines:
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.

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.

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

See also

- “download_cursor table event” on page 301
- “download_delete_cursor table event” on page 303
- “Synchronization of customers in the Contact sample” [MobiLink - Getting Started]
- “Synchronization of contacts in the Contact sample” [MobiLink - Getting Started]

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. See “Primary key pools” on page 112.

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 “Implementing timestamp-based downloads” on page 99.

The download_cursor script for this table is as follows.

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

```sql
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.

```sql
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
```

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
This script deletes all approved rows from the remote database.

**Partitioned foreign key tables**

The example in the previous section illustrates how to partition tables based on a criterion in some other table. See “Partitions with overlaps” on page 106.

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 foreign key 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 “Synchronization of 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.

**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].

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:

- “Composite keys”
- “UUIDs”
- “GLOBAL AUTOINCREMENT”
- “Primary key pools”

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 also

- “Remote IDs” [MobiLink - Client Administration]
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:

```sql
CREATE TABLE customer (
  cust_key UNIQUEIDENTIFIER NOT NULL DEFAULT NEWID(),
  rep_key VARCHAR(5),
  PRIMARY KEY(cust_key))
```

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 111.

Using GLOBAL AUTOINCREMENT columns

You can set the default column value to be GLOBAL AUTOINCREMENT to maintain unique values.
Prerequisites

There are no prerequisites for this task.

Context and remarks

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 111.

Task

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 “DEFAULT GLOBAL AUTOINCREMENT” on page 110.

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 “Global database IDs” on page 111.

Results

The column is set as a GLOBAL AUTOINCREMENT column.

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.

```sql
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]” [UltraLite - Database Management and Reference]

Global database IDs

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.

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.

In UltraLite, you set the global ID of a database by setting the `global_id` option.

See also
- “`global_database_id` option” [SQL Anywhere Server - Database Administration]
- “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 “Primary key pools” on page 112.

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 “Trigger conditions for events” [SQL Anywhere Server - Database Administration].

See also

● “Global database IDs” on page 111
● 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.

\[
\text{SET OPTION PUBLIC.global_database_id = 20}
\]

If the partition size for a particular column is 5000, default values for this database are selected from the range 100001-105000.

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.

Following is an overview of how 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 “Implementing timestamp-based downloads” on page 99.

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 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

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;
}
```
Conflict handling

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 can arise during the upload of rows to the consolidated database and 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.

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.

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:

\[
\text{current consolidated} - \left( \text{old remote} - \text{new remote} \right) \\
\rightarrow 7 - (10-6) = 3
\]

For other examples of how to handle conflicts, see:

● “Synchronization of products in the Contact sample” [MobiLink - Getting Started]
Conflict detection

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.

The MobiLink server detects conflicts only if an upload_fetch or upload_fetch_column_conflict script is applied. When you use upload_fetch, conflicting updates are flagged as a conflict. When you use upload_fetch_column_conflict, only conflicts on updates to the same column are flagged.

You can also set up arbitrary conflict detection and resolution using a stored procedure for upload_update. Conflict detection and resolution is completely controlled by the script so MobiLink does not trigger a conflict.

Conflict detection 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 (where pk1, pk2, ... are primary key columns and col1, col2, ... are non-primary columns):

\[
\text{SELECT } pk1, pk2, ... \text{col1, col2, ...} \\
\text{FROM } \text{table-name} \\
\text{WHERE } pk1 = \{ml r.pk1} \text{ AND } pk2 = \{ml r.pk2} \text{ ...} \\
\text{AND } \text{col1} = \{ml o.col1} \text{ AND } \text{col2} = \{ml o.col2} \text{ ...}
\]
Note
This method of conflict detection cannot 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 403.

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 database are updated in the consolidated.

See “upload_fetch_column_conflict table event” on page 405.

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 “Conflict resolution with resolve_conflict scripts” on page 118.

Conflict detection with upload_update scripts
You do not define scripts for upload_fetch, upload_fetch_column_conflict, 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 “Conflict resolution with upload_update scripts” on page 120.

Conflict resolution
You have several options for resolving conflicts:

● Resolve conflicts as they occur using temporary or permanent tables and a resolve_conflict script.
  See “Conflict resolution with resolve_conflict scripts” on page 118.

● Resolve conflicts as they occur using an upload_update script.
  See “Conflict resolution with upload_update scripts” on page 120.

● Resolve all conflicts at once using a table's end_upload script.
  See “end_upload table event” on page 331.

Conflict resolution 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 411.

● 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 409.

● 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.
See also

- “resolve_conflict table event” on page 385

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.

<table>
<thead>
<tr>
<th>Event</th>
<th>Script</th>
</tr>
</thead>
<tbody>
<tr>
<td>begin_synchronization</td>
<td>CREATE TABLE #contact_new(</td>
</tr>
<tr>
<td></td>
<td>id INTEGER,</td>
</tr>
<tr>
<td></td>
<td>location CHAR(36),</td>
</tr>
<tr>
<td></td>
<td>contact_date DATE);</td>
</tr>
<tr>
<td></td>
<td>CREATE TABLE #contact_old(</td>
</tr>
<tr>
<td></td>
<td>id INTEGER,</td>
</tr>
<tr>
<td></td>
<td>location CHAR(36),</td>
</tr>
<tr>
<td></td>
<td>contact_date DATE)</td>
</tr>
<tr>
<td>upload_fetch</td>
<td>SELECT id, location, contact_date</td>
</tr>
<tr>
<td></td>
<td>FROM contact</td>
</tr>
<tr>
<td></td>
<td>WHERE id = {ml r.id}</td>
</tr>
<tr>
<td>upload_old_row_insert</td>
<td>INSERT INTO #contact_new(   id, location,</td>
</tr>
<tr>
<td></td>
<td>contact_date )</td>
</tr>
<tr>
<td></td>
<td>VALUES ( {ml r.id}, {ml r.location}, {ml r.contact_date} )</td>
</tr>
<tr>
<td>upload_new_row_insert</td>
<td>INSERT INTO #contact_old(    id, location,</td>
</tr>
<tr>
<td></td>
<td>contact_date )</td>
</tr>
<tr>
<td></td>
<td>VALUES ( {ml r.id}, {ml r.location}, {ml r.contact_date} )</td>
</tr>
<tr>
<td>resolve_conflict</td>
<td>CALL MLResolveContactConflict( )</td>
</tr>
<tr>
<td>end_synchronization</td>
<td>DROP TABLE #contact_new;</td>
</tr>
<tr>
<td></td>
<td>DROP TABLE #contact_old</td>
</tr>
</tbody>
</table>

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
```
Conflict resolution 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:

```sql
{CALL UpdateProduct(  
    {ml o.id}, {ml o.name}, {ml o.desc}, {ml r.name}, {ml r.desc}  
)  }
```

The UpdateProduct stored procedure could be:

```sql
CREATE PROCEDURE UpdateProduct(  
    @id INTEGER,  
    @preName VARCHAR(20),  
    @preDesc VARCHAR(200),  
    @postId INTEGER,  
    @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 also

- “Conflict detection with upload_update scripts”
- “upload_update table event”
- “Conflict resolution with resolve_conflict scripts”
- “COALESCE function [Miscellaneous]” [SQL Anywhere Server - SQL Reference]
- `@@rowcount` in “Global variables” [SQL Anywhere Server - SQL Reference]
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.

```sql
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)
)
```

```sql
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 database 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 synchronization techniques
    END
```

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0 121
// 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 database 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;

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, click 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 also
- “download_delete_cursor scripts” on page 244

**Temporarily stopping the synchronization of deletes**

Use the STOP SYNCHRONIZATION DELETE statement to temporarily stop the automatic logging of changes to tables or columns that are part of a publication with a synchronization subscription, which are normally uploaded to the consolidated database during the next synchronization.

**Prerequisites**

There are no prerequisites for this task.

**Context and remarks**

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.

**Task**

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
   ```

**Results**

The deleted rows are not sent up to the MobiLink server and are not deleted from the consolidated database.
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].

Resumption of failed downloads

Download failure is caused by a communication error during the download or a remote user canceling 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 mlsrv16 option” on page 54.
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.

  - Embedded SQL “ULRollbackPartialDownload method [UltraLite Embedded SQL]” [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 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 cannot be used with resumable downloads. See “Resumption of failed downloads” on page 124.

See also

- “publication_nonblocking_download_ack connection event” on page 375
- “nonblocking_download_ack connection event” on page 369
- dbmlsync: “SendDownloadAck (sa) extended option” [MobiLink - Client Administration]
- UltraLite: “Send Download Acknowledgement synchronization parameter” [UltraLite - Database Management and Reference]

**Result sets from stored procedure calls**

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 CURRENT_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.):
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.

```sql
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 SQL Anywhere 16 - 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.

```sql
create or replace procedure DownloadMyTable(
    v_last_dl_ts IN TIMESTAMP,
    v_user_name IN VARCHAR,
    v_ref_crsr OUT SYS_REF_CURSOR )
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:

```sql
CALL ml_add_table_script(
    'v1',
    'MyTable',
    'download_cursor',
    '(CALL DownloadMyTable(
        {ml s.last_table_download}, {ml s.username} ))')
);
```

For Oracle, 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.

```sql
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;

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 database should be sent as a separate transaction.

● Add a mapping table, mapping employees to managers, so the order of transactions in the formerly self-referencing table no longer matters.

See also

● “-tu dbmlsync option” [MobiLink - Client Administration]

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 “How to enable snapshot isolation” [SQL Anywhere Server - SQL Usage].

You can enable snapshot isolation for upload with the mlsrv16 -esu option, and disable snapshot isolation with the mlsrv16 -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 mlsrv16 option” on page 56 and “-dsd mlsrv16 option” on page 54.

**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 mlsrv16 -esu option, and disable snapshot isolation with the mlsrv16 -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 mlsrv16 option” on page 56 and “-dsd mlsrv16 option” on page 54.

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 mlsrv16 option” on page 55.

See also

- “-dsd mlsrv16 option” on page 54
- “-esu mlsrv16 option” on page 56
- “-dt mlsrv16 option” on page 55
- “The synchronization process” [MobiLink - Getting Started]
- “Isolation levels and consistency” [SQL Anywhere Server - SQL Usage]
- “Snapshot isolation” [SQL Anywhere Server - SQL Usage]

MobiLink performance

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 656.

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 137.

For information about the transaction structure of synchronization, see “Transactions in the synchronization process” [MobiLink - Getting Started].

**Use multi-threaded network processing**
The MobiLink server supports multiple network worker threads processing its network streams concurrently. Having multiple network worker threads can improve performance, particularly when using CPU-intensive network stream options, like encryption or compression, with either large synchronizations or many small synchronizations. Each request in the system can be active on one network stream thread, at most.

Use the -wn mlsrv16 option to set stream threads. See “-wn mlsrv16 option” on page 75.
Use an optimal number of database worker threads
You can either choose a fixed number of MobiLink database worker threads or let the MobiLink server automatically adjust the number. If you use a fixed number by not using the -wm option, you need to experiment with different values for the -w option to determine the smallest number that gives you optimum throughput. With automatic adjustment, you specify the maximum number of database worker threads via the -wm option, and -w is used to specify the minimum and initial number. If the -w option is not used, the default is 5. For example, if you use -wm 50 and do not use the -w option, the MobiLink server periodically adjusts the number of active database worker threads to numbers in the range from 5 to 50.

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.

See also
● “Number of database worker threads” on page 138
● “Automatic adjustment of database worker threads” on page 133
● “-wm mlsrv16 option” on page 84
● “-w mlsrv16 option” on page 83
● “-wu mlsrv16 option” on page 84

Automatic adjustment of database worker threads
The MobiLink server can automatically adjust the number of database worker threads based on load, with the goal of maximizing throughput. To enable this automatic adjustment, use the -w mlsrv16 option to set the initial number of concurrent database worker threads and the -wm mlsrv16 option to set the maximum number of concurrent database worker threads. The MobiLink server monitors system performance and adjusts the number of database worker threads up and down within the parameters set by the -w and -wm options, depending on the requirements at any given time.

The value set with the -wu mlsrv16 option, which represents the maximum number of database worker threads that can apply uploads to the consolidated database simultaneously, is not automatically adjusted. If the -wu option is not specified, uploads may be applied on any or all database worker threads, and the number of database worker threads is variable. When the MobiLink server increases the number of database worker threads in an attempt to increase throughput, contention may increase in the short term until it is detected and the thread-count is eventually decreased.

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 mlsrv16 option” on page 78.

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 mlsrv16 -cn option.

See “MobiLink database connections” on page 139 and “-cn mlsrv16 option” on page 51.

**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.
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 Profiler. The MobiLink Profiler 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 Profiler” on page 176.

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 -1305 (SQLE_MOBILINK_COMMUNICATIONS_ERROR). The client application should handle this error and try to connect again in a few minutes.

See:

- “-sm mlsrv16 option” on page 74
- “-nc mlsrv16 option” on page 59
- “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 database 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.

**MobiLink tuning 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.

Use the -w mlsrv16 option to set the number of database worker threads. You can also use the -wm mlsrv16 option to let MobiLink server automatically adjust the number of database worker threads for the best throughput—assuming the number lies between the current -w and -wm settings. However, while -wm is convenient it may not work in all cases. As always, testing is vital to determine the optimum number of database worker threads.

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 mlsrv16 option” on page 50.
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 mlsrv16 option” on page 74.

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, so 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.

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.

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.

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.

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.

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.

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.

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.

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.
An example of tuning MobiLink for two script versions is given in the following command line:

```bash
mlsrv16 -c "DSN=SQL Anywhere 16 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 mlsrv16 option” on page 51.

### MobiLink performance monitoring

There are a variety of tools available to help you monitor the performance of your synchronizations.

The MobiLink Profiler is a graphical tool for monitoring synchronizations. It allows you to see the time taken by every aspect of the synchronization.

See “MobiLink Profiler” on page 176.

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 305
- “download_statistics table event” on page 308
- “synchronization_statistics connection event” on page 388
- “synchronization_statistics table event” on page 392
- “time_statistics connection event” on page 395
- “time_statistics table event” on page 398
- “upload_statistics connection event” on page 413
- “upload_statistics table event” on page 417

### 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 16 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 16 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 16 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.

A sample MobiLink project is provided in %SQLANYSAMP16%\MobiLink\CustDB\project.mlp.

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_computename_UUID, where computename is the host name of the computer that the Agent is running on and UUID is a universally unique identifier.

It is highly recommended that case not be used to differentiate between Agent IDs. For example, do not create Agent ID Agent_XYZ and agent_xyz as different Agent IDs. When the consolidated database is case-insensitive, this recommendation is a requirement. When the consolidated database is case-sensitive, this recommendation is not enforced.

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 remote task

A deployed remote 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 16\MobiLink Agent` on Windows and `My Device\Application Data\SQLAny16\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 are 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.

When you add a consolidated database to a project, either with the Create Project Wizard or the Add Consolidated Database Wizard, the wizard automatically checks if there are any remote schema names defined in the consolidated database that are not already in the project. If there are, you are asked to import them.

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 following steps outline the procedures to set up central administration of remote databases on the server and client.

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 “Remote tasks” on page 155.

**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 “MobiLink Agent on the client device” on page 145.

**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 following diagram shows the flow of communication to and from the Agent.
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.
See also

- “mlagent command” on page 146

mlagent command

Runs the MobiLink Agent on the client device, either in configuration mode or normal mode.

Syntax

`mlagent [ options ]`

To run in configuration mode, specify -c or -cr on the mlagent command line.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@data</td>
<td>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 “Configuration files” [SQL Anywhere Server - Database Administration].</td>
</tr>
<tr>
<td>-c</td>
<td>Set the configuration options, only updating those options that differ from the current options, then stop the Agent.</td>
</tr>
<tr>
<td>-cr</td>
<td>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.</td>
</tr>
<tr>
<td>-a agentid</td>
<td>Valid only with -c or -cr. Specify the ID of this Agent.</td>
</tr>
</tbody>
</table>

If the -a option is not specified, the default is `Agent_computername_UUID`, where `computername` is the host name of the computer that the Agent is running on and `UUID` is a universally unique identifier.

It is highly recommended that case not be used to differentiate between Agent IDs. For example, do not create Agent ID `Agent_XYZ` and `agent_xyz` as different Agent IDs. When the consolidated database is case-insensitive, this recommendation is a requirement. When the consolidated database is case-sensitive, this recommendation is not enforced.

<p>| -db database location | Valid only with -c or -cr. Specify the path where remote databases are stored on the client device. |</p>
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-x protocol[protocol-options] ... protocol : tcpip</td>
<td>Valid only with -c or -cr. Specify the MobiLink client network protocol options. These options determine how the Agent connects to the MobiLink server when synchronizing the agent database.</td>
</tr>
<tr>
<td>-ap parameters</td>
<td>Specify the MobiLink authentication parameters used when synchronizing the agent database.</td>
</tr>
<tr>
<td>-ek key</td>
<td>Specify the encryption key used to access the agent database.</td>
</tr>
<tr>
<td>-n name</td>
<td>Specify the name of the agent database. The default is taskdb.</td>
</tr>
<tr>
<td>-o file</td>
<td>Log output to the specified file.</td>
</tr>
<tr>
<td>-on size</td>
<td>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.</td>
</tr>
<tr>
<td>-os size</td>
<td>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.</td>
</tr>
<tr>
<td>-ot file</td>
<td>Truncate the file and log output messages to it.</td>
</tr>
<tr>
<td>-p password</td>
<td>Specify the MobiLink password used to synchronize the agent database.</td>
</tr>
<tr>
<td>-pi</td>
<td>Test whether the Agent can synchronize. This option 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. For more information about ping synchronizations, see “-pi dbmlsync option” [MobiLink - Client Administration]. 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.</td>
</tr>
<tr>
<td>-q</td>
<td>Run in a minimized window.</td>
</tr>
</tbody>
</table>
### Option

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-qi</td>
<td>Do not display tray icon or window.</td>
</tr>
<tr>
<td>-u user</td>
<td>Specify the MobiLink user name used to synchronize the agent database.</td>
</tr>
<tr>
<td>-v level</td>
<td>Specify the output verbosity level from 0-9. The default is 3.</td>
</tr>
</tbody>
</table>

### 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:

```bash
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:

```bash
mlagent
```

### Interactive configuration of the MobiLink Agent

The MobiLink Agent can be configured using a configuration window as well as through the command line. The configuration window appears automatically if the Agent is run and there is not enough information for the Agent database to operate properly. Alternatively, you can display the configuration window using the menu on the MobiLink Agent window. For Windows, click the System menu in the top left corner of the MobiLink Agent window and click Configure. For Windows Mobile, select the Configure item on the menu bar.
The following tables shows how the fields on the configuration window relate to the MobiLink Agent configuration options.

<table>
<thead>
<tr>
<th>Configuration window field name</th>
<th>Equivalent mlagent configuration option</th>
</tr>
</thead>
<tbody>
<tr>
<td>Agent ID</td>
<td>-a option for mlagent</td>
</tr>
<tr>
<td>User</td>
<td>-u option for mlagent</td>
</tr>
<tr>
<td>Password</td>
<td>-p option for mlagent</td>
</tr>
<tr>
<td>Authentication Parameters</td>
<td>-ap option for mlagent</td>
</tr>
<tr>
<td>MobiLink Client Network Protocol Options</td>
<td>-x option for mlagent</td>
</tr>
<tr>
<td>Remote Database Location</td>
<td>-db option for mlagent</td>
</tr>
</tbody>
</table>

For more information about configuration options for the MobiLink Agent, see “MobiLink Agent on the client device” on page 145.
Encryption

If the Agent database does not exist, a message appears asking if a default encryption key should be used. The default key is hard to guess, but could be discovered since it is hard-coded in the Agent. If you choose not to use the default encryption key, the following window appears so you can enter and verify the encryption key:

If the Agent database exists but the MobiLink Agent cannot connect to it because a non-default encryption key was not specified with the -ek option, the following window appears so you can enter the key:

MobiLink Agent Stop utility

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 ]
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-n name</td>
<td>The name of the Agent to stop. If -n is not specified, all Agents on the device are stopped.</td>
</tr>
</tbody>
</table>

See also

- “mlagent command” on page 146
- “SQL Anywhere MobiLink client deployment” on page 728
- “UltraLite MobiLink client deployment” on page 731
MobiLink 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.

You must have a remote schema name defined before you can create an Agent in Sybase Central.

See:

- “Adding a remote schema name” on page 151
- “Importing remote schema names” on page 151
- “Adding an Agent” on page 152
- “Agent properties” on page 153
- “Viewing or changing Agent properties” on page 153
- “Adding managed remote databases” on page 154
- “Adding a group” on page 154

Adding a remote schema name

Use the Create Remote Schema Name Wizard to add a remote schema name.

Prerequisites

There are no prerequisites for this task.

Context and remarks

A remote schema name must be defined before you can create an Agent in Sybase Central.

Task

1. Double-click the MobiLink project.
   
   Double-click Remote Schema Names and then click New Remote Schema Name.

2. Follow the instructions in the Create Remote Schema Name Wizard and click Finish.

Results

The remote schema name is added.

Importing remote schema names

Schema names can be imported from another database.

Prerequisites

There are no prerequisites for this task.
Task

1. Double-click the MobiLink project.
2. Right-click Remote Schema Names and click Import.
3. Choose the database from which you want to import the remote schema names from the displayed list of consolidated databases and click OK.

Results

Any remote schema names in the selected database that are not already in the MobiLink project are imported.

Adding an Agent

Add an Agent to use central administration of remote databases.

Prerequisites

A remote schema name must be defined before you can create an Agent in Sybase Central.

Task

1. Double-click the MobiLink project.
2. Double-click Consolidated Databases.
3. Double-click Agents and then click File » New » Agent.
4. Follow the steps in the Create MobiLink Agent Wizard.

Results

The Agent is created.

Next

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
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** The synchronization interval controls how frequently the Agent synchronizes its agent database.

- **Administration Polling Interval** The administration polling interval determines how frequently the Agent checks for requests from the server for it to synchronize or perform other actions.

- **MobiLink Client Network Protocol Options** 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.

- **Connection Strings for Managed Databases** The connection string(s) that the MobiLink Agent can use to connect to the remote database(s).

Viewing or changing Agent properties

Agent properties can be viewed or updated.

**Prerequisites**

There are no prerequisites for this task.

**Task**

1. Double-click the consolidated database.

2. Double-click **Agents**, right-click the Agent you want to work with and click **Properties**.

3. If necessary, make changes to the properties and click **Apply**.

**Results**

The Agent properties are updated.
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.

Prerequisites

There are no prerequisites for this task.

Task

1. Double-click the consolidated database.

2. Double-click the Agent you want to worth with and click Add Managed Remote Database.

3. The available remote databases are represented by remote schema names. Choose a Remote Schema Name from the dropdown list.

4. 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.

Results

The database is added.

Adding a group

A group is a collection of MobiLink Agents.

Prerequisites

You must have one of more Agents defined before you can create a group.

Task

1. Ensure that your consolidated databases are running.

2. Double-click the MobiLink project.

3. Double-click the project name and click New » Group.

4. Follow the instructions in the Group Wizard and click Finish.

Results

The group is created.
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_16 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 “Consolidated database setup” on page 2.

At least one of the following must be defined to have a secure system:

- “authenticate_user connection event”
- “authenticate_user_hashed connection event”
- “authenticate_parameters connection event”

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_16', 'authenticate_user', '<DEFINE YOUR AUTHENTICATION LOGIC>' )
  ```

- Use Connection Scripts in Sybase Central. You can use the ml_ra_agent_16 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 233.

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 16 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 166.

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 canceled, initiated (SIRT), reactivated and assigned to new recipients.

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 16 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 want 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 Task (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 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 166.

- **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.

- **Random delay interval** If a given task sends results to the MobiLink server after execution or causes a remote database to synchronize with the server, and it is triggered simultaneously across a
large number of remotes, then setting a random delay for the task uniformly distributes the synchronization workload for the server over a configurable period of time.

A remote task can have a random delay interval, which is an interval $N$, in seconds, with which each agent generates a random number of seconds in $[0, N)$ to delay each task execution. If the task is a scheduled task, the random delay is generated before the first task execution, and used for each execution. The task is executed at the scheduled times, offset by the random delay. This ensures that the deltas of the task execution times are consistent with the schedule.

It is not recommended that the random delay interval be larger than smallest delta time of a scheduled task. If the task is an on demand task, meaning it is initiated by the server, the random delay is generated and used to delay the execution each time the task is initiated. If the task is a run on receipt task, the random delay is generated and used to delay the execution at the first and only time the task is executed.

See also
- “Remote task logic” on page 158
- “Creating a remote task” on page 159
- “Editing a remote task” on page 160
- “Deploying a remote task” on page 160
- “Deployed remote tasks” on page 161

Remote task logic

The following is an outline of the logic used to execute a remote task.

```c
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;
      }
    }
  }
```
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.

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.

Prerequisites

There are no prerequisites for this task.

Context and remarks

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.

Task

1. Double-click the MobiLink project.


3. 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.

4. Follow the procedure to add one or more commands to the task. See “Adding a command to a remote task” on page 171.

Results

The task is created.
After the task has been created, it can be deployed to the consolidated, and then assigned to recipients (Agents). See “Deploying a remote task” on page 160.

Editing a remote task

Remote task properties can be edited.

Prerequisites

There are no prerequisites for this task.

Task

1. Double-click the MobiLink project.
2. Click Properties.
3. Edit the remote task properties.
4. When you are finished editing the remote task properties, click Apply.

Results

The remote task properties are updated.

Deploying a remote task

When you are ready to add the task to the system, the task needs to be deployed.

Prerequisites

There are no prerequisites for this task.

Context and remarks

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.
Task

A remote task must contain at least one command before it can be deployed. For information about commands, see “Commands” on page 166 and “Adding a command to a remote task” on page 171.

1. Double-click the MobiLink project.
2. Right-click the remote task you want to deploy and click Deploy.
3. Follow the instructions in the Deploy a Remote Task Wizard.
4. Click Finish.

Results

The remote task is deployed.

Next

Deployed tasks can be canceled, initiated, reactivated and have recipients added to them. See “Deployed remote tasks” on page 161.

Exporting a remote task

Remote tasks can be exported to a file.

Prerequisites

You must have a remote task defined.

Task

1. Double-click the MobiLink project.
2. Right-click the remote task you want to export and click Export.
3. Specify a name and location for the file. Click Save.

Results

The remote task is exported to the specified file.

Deployed remote tasks

You can still work with a remote task after it has been deployed. Deployed tasks can be canceled, initiated, reactivated and have recipients added to them.

You can find deployed remote tasks in Sybase Central in the following places:
To work with deployed remote tasks at an individual Agent level, ensure the consolidated database you are working with is expanded in the Folders view of the MobiLink 16 plug-in for Sybase Central. Expand Agents, click the Agent you want to work with then click the Tasks tab in the right pane. It contains a list of all remote tasks that have been deployed for the selected Agent.

To work with deployed remote tasks for all Agents, ensure the MobiLink project you are working with is selected in the left pane in Folders view, then right-click Remote Tasks and click Deployed Tasks. Deployed remote tasks are listed for all Agents.

See also
- “Canceling a deployed remote task for all Agents” on page 162
- “Canceling a deployed remote task for a single Agent” on page 162
- “Initiating a deployed remote task for all Agents” on page 163
- “Initiating a deployed remote task for a single Agent” on page 163
- “Reactivating a deployed remote task for a single Agent” on page 164
- “Adding recipients to a deployed remote task” on page 164

## Canceling a deployed remote task for all Agents

You may want to cancel a deployed remote task if it is no longer required, or if you are upgrading the remote schema and are deploying a new version of the task.

**Prerequisites**

There are no prerequisites for this task.

**Task**

1. Double-click the MobiLink project.

2. Click Remote Tasks » Deployed Tasks and select the deployed task you want to cancel.

3. Right-click the deployed task you want to cancel and click Cancel For All Recipients.

**Results**

The task is canceled.

## Canceling a deployed remote task for a single Agent

You may want to cancel a deployed remote task if it is no longer required, or if you are upgrading the remote schema and are deploying a new version of the task.

**Prerequisites**

There are no prerequisites for this task.
Task

1. Double-click the consolidated database.
2. Double-click **Agents** and click the Agent you want to work with.
3. In the right pane, click the **Tasks** tab.
4. Right-click the deployed task you want to cancel and click **Cancel**.

Results

Initiating a deployed remote task for all Agents

Initiating a deployed remote task causes the server to notify the Agent on the client that the task should be run right away.

Prerequisites

There are no prerequisites for this task.

Task

1. Double-click the MobiLink project.
2. Click **Remote Tasks » Deployed Tasks** and select the deployed task you want to initiate.
3. Right-click the deployed task you want to initiate and click **Initiate For All Recipient**.

Results

The task is initiated for all Agents.

Initiating a deployed remote task for a single Agent

Initiating a deployed remote task causes the server to notify the Agent on the client that the task should be run right away.

Prerequisites

There are no prerequisites for this task.

Task

1. Double-click the consolidated database you.
2. Double-click **Agents** and click the Agent you want to work with.
3. In the right pane, click the **Tasks** tab.

4. Right-click the deployed task you want to initiate and click **Initiate**.

**Results**

The task is initiated for the selected Agent.

---

**Reactivating a deployed remote task for a single Agent**

Some deployed remote tasks can be reactivated.

**Prerequisites**

There are no prerequisites for this task.

**Task**

1. Double-click the consolidated database.

2. Double-click **Agents** and click the Agent you want to worth with.

3. In the right pane, click the **Tasks** tab.

4. Right-click the deployed task you want to reactivate and click **Reactivate**.

**Results**

The deployed task is reactivated.

---

**Adding recipients to a deployed remote task**

You can add recipients to a deployed remote task. For example, you would need to add Agents to a deployed task if new Agents are created after a task has been deployed.

**Prerequisites**

There are no prerequisites for this task.

**Task**

1. Double-click the MobiLink project.

2. Click **Remote Tasks » Deployed Tasks** and select the deployed task you want to add recipients to.

3. Right-click the deployed task you want to add recipients to and click **Add Recipients**.

4. Choose Agents from the **Agents** list and click **Add** to add them to the **Recipients** list, or choose **Add All** to select all Agents.
5. Click **Ok**.

**Results**

The selected Agents are added as recipients.

---

**Server-initiated remote tasks (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. When an administrator chooses to initiate a remote task, the MobiLink server sends a message to the affected Agents instructing them to run the specified task. A task might already be scheduled to run at a particular time, however, it could still be a server-initiated if the administrator chooses to do so.

A SIRT can be initiated through Sybase Central by choosing **Initiate** for a particular Agent or **Initiate For All Recipients** on a deployed task. A SIRT can also be initiated using the `ml_ra_notify_task` system procedure on the MobiLink server.

**See also**

- “`ml_ra_notify_task` system procedure” on page 649

---

**Remote task notifier (RTNotifier)**

A notifier called the RTNotifier is built into the MobiLink server to keep track of SIRT requests. The RTNotifier checks the MobiLink system tables and if a SIRT has been initiated, when the MobiLink Agent polls the MobiLink server the RTNotifier sends the appropriate remote task information to the client and the remote task is run.

The RTNotifier runs by default. If you are not using central administration of remote tasks, you can disable the RTNotifier using the options below.

RTNotifier options are specified as option/value pairs that get inserted into the `ml_property` system table. The example below shows how to turn the RTNotifier off. In the example, the RTNotifier option is **enable** and the value is set to no. SIRT is the component name, and the **RTNotifier(RTNotifier1)** is the notifier name, but these two columns are for internal use only and should not be changed.

```sql
   call ml_add_property( 'SIRT', 'RTNotifier(RTNotifier1)', 'enable', 'no' );
```

The table below lists RTNotifier options that can be specified in the `ml_property` system table.

<table>
<thead>
<tr>
<th>Option</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>autoset_poll_every</td>
<td>{ yes</td>
<td>no }</td>
</tr>
<tr>
<td>Option</td>
<td>Value</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>enable</td>
<td>{ yes</td>
<td>no }</td>
</tr>
<tr>
<td>poll_every</td>
<td>time in seconds</td>
<td>Specifies how often, in seconds, the RTNotifier executes the request_cursor to re-populate the in-memory cache with remote task requests. If the value of this property is 2147483647, then the RTNotifier does not execute the request_cursor.</td>
</tr>
<tr>
<td>update_poll_every</td>
<td>time in seconds</td>
<td>Specifies how often, in seconds, the RTNotifier should check for updates to their properties.</td>
</tr>
<tr>
<td>request_cursor</td>
<td></td>
<td>This option is for internal use only. Specifies a query used to retrieve the remote task requests from the consolidated database.</td>
</tr>
</tbody>
</table>

**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 will 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.

By default, a task fails if the file to be deleted does not exist. To allow a task to succeed even if the file to be deleted does not exist, clear the **Fail If The File Does Not Exist** option on the **Commands** property page for the delete file command in the MobiLink 16 plug-in for Sybase Central.

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 cannot 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 mlsrv16 option” on page 58.

- **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 command

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:

  ```sql
  SELECT * FROM systable
  GO
  CREATE PROCEDURE p1()
  BEGIN
      CREATE TABLE t1( pk INTEGER PRIMARY KEY );
  ```
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 cannot 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 server 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-writing 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, 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 172.

Command usage

Use one of the following methods to add a command to a task:

- In the Folders view in the left pane, right-click the task and click 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 an existing command and click Add Command.
- Right-click in the whitespace under the existing commands and click Add Command.
- Select the task in the Folders view in the left pane. From the File menu click Add Command.

See “Adding a command to a remote task” on page 171.

Adding a command to a remote task

A remote task cannot be deployed until it contains at least one command.

Prerequisites

There are no prerequisites for this task.

Task

1. Double-click the MobiLink project.

2. Double-click Remote Tasks, right-click the remote task you want to work with, and click 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 about the available commands, see “Commands” on page 166.

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.

**Results**

Result

**Next**

The task can be deployed. See “Deploying a remote task” on page 160.

**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:

<table>
<thead>
<tr>
<th>Variable</th>
<th>Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>{agent_db}</td>
<td>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.</td>
</tr>
<tr>
<td>{agent_id}</td>
<td>The Agent ID.</td>
</tr>
<tr>
<td>{agent_log}</td>
<td>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.</td>
</tr>
<tr>
<td>{battery_level}</td>
<td>The battery level for the remote device. The range is zero to one hundred (0-100).</td>
</tr>
<tr>
<td>{db_location}</td>
<td>The Agent's remote database directory, as specified by the <code>mlagent -db</code> option.</td>
</tr>
<tr>
<td>{is_on_ac_power}</td>
<td>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.</td>
</tr>
<tr>
<td>{is_online}</td>
<td>This variable evaluates to 1 (true) if and only if the client device is connected to a network such that there is a route to the IP address of the MobiLink server. The host computer may be offline and the variable will still evaluate to 1 (true).</td>
</tr>
<tr>
<td>Variable</td>
<td>Replacement</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>{ml_password}</code></td>
<td>The MobiLink password (pwd) being used by the Agent to synchronize with the agent database.</td>
</tr>
<tr>
<td><code>{ml_stream}</code></td>
<td>The protocol parameters for connecting to the MobiLink server. For example, HTTP({host=Sybase.com;port=9376}}</td>
</tr>
<tr>
<td><code>{ml_username}</code></td>
<td>The MobiLink user name (uid) being used by the Agent to synchronize with the agent database.</td>
</tr>
<tr>
<td><code>{network_conn_name}</code></td>
<td>Evaluates to the name of the network connection that is used by the Agent for communication with the MobiLink server. If there is no network connection that can be used by the Agent to communicate with the MobiLink server, this variable evaluates to ?.</td>
</tr>
<tr>
<td><code>{remote_id}</code></td>
<td>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.</td>
</tr>
<tr>
<td><code>{rows_to_upload}</code></td>
<td>For UltraLite databases only. The number of rows in the remote database that will be uploaded if a full synchronization is done.</td>
</tr>
</tbody>
</table>

**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 16 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”
- “ml_ra_get_agent_ids system procedure”
- “ml_ra_get_agent_properties system procedure”
- “ml_ra_get_latest_event_id system procedure”
- “ml_ra_get_orphan_taskdbs system procedure”
- “ml_ra_get_remote_ids system procedure”
- “ml_ra_get_task_results system procedure”
- “ml_ra_get_task_status system procedure”
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 16 plug-in for Sybase Central. However, the following tasks can only be done using the MobiLink 16 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 databases:

- `ml_ra_add_agent_id` system procedure
- `ml_ra_assign_task` system procedure
- `ml_ra_cancel_notification` system procedure
- `ml_ra_cancel_task_instance` system procedure
- `ml_ra_clone_agent_properties` system procedure
- `ml_ra_delete_agent_id` system procedure
- `ml_ra_delete_events_before` system procedure
- `ml_ra_delete_remote_id` system procedure
- `ml_ra_delete_task` system procedure
- `ml_ra_get_agent_events` system procedure
- `ml_ra_get_agent_ids` system procedure
- `ml_ra_get_agent_properties` system procedure
- `ml_ra_get_latest_event_id` system procedure
- `ml_ra_get_orphan_taskdbs` system procedure
- `ml_ra_get_remote_ids` system procedure
- `ml_ra_get_task_results` system procedure
- `ml_ra_get_task_status` system procedure
- `ml_ra_manage_remote_db` system procedure
- `ml_ra_notify_agent_sync` system procedure
- `ml_ra_reassign_taskdb` system procedure
- `ml_ra_set_agent_property` system procedure
- `ml_ra_unmanage_remote_db` system procedure

Deployment and configuration

The following section provide information about deployment and configuration. For a list of files required to deploy the MobiLink Agent, see “SQL Anywhere MobiLink client deployment” on page 728 and “UltraLite MobiLink client deployment” on page 731.
Agent deployment considerations

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 so applications 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 “UltraLite data management components for Windows Mobile” [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 “MobiLink Agent Stop utility” on page 150.

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 databases.

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 computer 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 about 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 “MobiLink Agent on the client device” on page 145.

**MobiLink Profiler**

The MobiLink Profiler is a MobiLink administration tool that provides you with detailed information about the performance of your synchronizations, enabling you to analyze bottlenecks and maximize performance. You can use the SQL Anywhere Monitor for basic performance information and use the MobiLink Profiler to get lower level details, down to the event level, about synchronizations.

Synchronization data from your profiling session is saved in a profiling database file that is created in your data directory with a default file name, user, and password. You can specify a different profiling database on the **General** tab of the **Options** window.

When you start the MobiLink Profiler and connect it to a MobiLink server, the MobiLink Profiler begins to collect statistical information about all synchronizations that occur in that profiling session. The MobiLink Profiler continues to collect data until you end the profiling session or shut down the MobiLink server. You can view the data in tabular or graphical form in the MobiLink Profiler interface.

MobiLink Profiler output allows you to see a wide variety of information about your synchronizations. For example, you can quickly identify synchronizations or events that result in errors or that meet other criteria that you specify. You can identify possible contention in synchronization scripts by checking whether 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). You can also identify events for which the MobiLink server detected blocking.

It is recommended that the Profiler be used primarily in a development environment to test performance before deploying to a production system.

**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 Profiler.

For more information about the SQL Anywhere Monitor, see “SQL Anywhere Monitor” [SQL Anywhere Server - Database Administration].

**Starting the MobiLink Profiler (administration tools)**

You can have multiple instances of the MobiLink Profiler running for each MobiLink server. However, it is recommended that you only run one MobiLink Profiler instance per MobiLink server.

**Prerequisites**

For new profiling sessions, start your consolidated database and MobiLink server, if they are not already running.

**Context and remarks**

**Note**
The version of the MobiLink Profiler must match the version of the MobiLink Server you are using.

**Task**

- Click Start » Programs » SQL Anywhere 16 » Administration Tools » MobiLink Profiler.

**Results**
The MobiLink Profiler is started.

**Next**

Begin a profiling session to start collecting data. See “Starting a profiling session” on page 178.

**MobiLink Profiler (mlprof) on the command line**

Command line options allow you to have the MobiLink Profiler open and connect to a MobiLink server on startup. This is useful for automated, unattended profiling of a test session.

Use the following syntax:

```
miprof [ options ]
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-c</td>
<td>Closes the MobiLink Profiler at the end of the profiling session.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>-p password</td>
<td>The password for the MobiLink user.</td>
</tr>
<tr>
<td>-r</td>
<td>Recreates the profiling database.</td>
</tr>
<tr>
<td></td>
<td>Use this option to remove all previous profiling sessions, or if there is a</td>
</tr>
<tr>
<td></td>
<td>problem with the profiling database schema.</td>
</tr>
<tr>
<td>-u ml_username</td>
<td>The MobiLink user. This option is required to begin a profiling session</td>
</tr>
<tr>
<td></td>
<td>initiated from the command line.</td>
</tr>
<tr>
<td>-x {tcpip</td>
<td>ts</td>
</tr>
<tr>
<td></td>
<td>The keyword=value pairs can be the host, port, and additional network</td>
</tr>
<tr>
<td></td>
<td>parameters. This option is required to begin a profiling session initiated</td>
</tr>
<tr>
<td></td>
<td>from the command line.</td>
</tr>
</tbody>
</table>

You can type `mlprof -?` to view the mlprof syntax.

## Starting a profiling session

Starting a MobiLink Profiler session begins the collection of data and saves the data to a profiling database.

### Prerequisites

Start your consolidated database and MobiLink server, if they are not already running.

### Task

1. From the MobiLink Profiler, click **File » Begin Profiling Session**. This starts the collection of data.

2. A MobiLink Profiler connection starts like a synchronization connection to the MobiLink server. For all MobiLink Profiler sessions, the script version is set to `for_ML_Monitor_only`.

The **Connect To MobiLink Server** window should be completed as follows:

- **User**  Type the name of the MobiLink user for the connection. A user name must be supplied, but if you started the MobiLink server with `-zu+`, then it does not matter which MobiLink user you supply because unrecognized MobiLink user names are added automatically to the ml_user table upon synchronization.

- **Password**  Type a password for the connection. This must be the correct password for the MobiLink user you specify. Leave this field blank if the MobiLink user does not have a password.

- **Host**  The network name or IP address of the computer where the MobiLink server is running. By default, the host is the computer where the MobiLink Profiler is running. You can use `localhost` if the MobiLink server is running on the same computer as the MobiLink Profiler.
● **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. 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 Profiler. For more information about security, see “MobiLink client/server communications encryption” [SQL Anywhere Server - Database Administration].

### Note
Separately licensed component required.

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

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

● **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 that 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. Click the **Pause** button to pause automatic scrolling of the Chart and Utilization graph information.

### Results

The MobiLink Profiler starts collecting data and the profiling data appears as it is collected.

### Ending a profiling session

Ending a MobiLink Profiler session stops the collection of data but keeps the MobiLink Profiler running so that you can view your data or start a new profiling session.

### Prerequisites

There are no prerequisites for this task.
Task

1. Click File » End Profiling Session. This stops the collection of data and disconnects the Profiler from the MobiLink server.

   You can also stop collecting data by shutting down the MobiLink server or by closing the MobiLink Profiler.

2. When you are ready to close the MobiLink Profiler, click File » Close.

Results

The MobiLink Profiler stops collecting profiling data.

Opening or deleting a previous profiling session

Choose a previous MobiLink Profiler session to open or delete from the Open MobiLink Profiler Session window.

Prerequisites

There must be previous Profiler sessions in the profiling database.

Task

1. From the MobiLink Profiler, click File » Open Session. Previous profiling sessions in the profiling database are displayed.

2. Select the profiling session you want to open or delete.

3. Click OK to open the selected session, or click Delete to remove the selected session from the profiling database.

Results

If you clicked OK, the data from the selected session is displayed.

If you clicked Delete, the selected session is deleted from the profiling database and is no longer listed.

Next

You can review the data if you opened a previous Profiler session.

The profiling database

When the MobiLink Profiler first starts, it creates a profiling database with a default file name, user and password. The profiling database is stored in the MProfiler16 folder of your documents directory. If you do not want to use the default user and password for the profiling database, change the database location to an existing SQL Anywhere database that you want to use, or copy that database to the default location.
When you next start the MobiLink Profiler, you can enter the user ID and password for that database and it becomes the profiling database. Connection information is saved upon successful connection.

Once connected to the profiling database, if an incompatible schema is detected, you are asked if you want to recreate the profiling database schema and then the MobiLink Profiler closes. If you chose to recreate the profiling database, then when the Profiler is restarted, it recreates the schema, which also deletes any previous profiling session data. Data could be lost unless you back it up before restarting the MobiLink Profiler.

To determine the name and location of the profiling database, go to the General page of the Options window.

**MobiLink Profiler interface**

The MobiLink Profiler has the following panes:

- **Details Table**  
  Details Table is the top pane, when enabled. It is a spreadsheet that, by default, shows the total time taken by each synchronization, with a breakdown showing the amount of time taken by each phase of the synchronization.

  See “Details Table pane” on page 182.

- **Utilization Graph**  
  Utilization Graph is the second pane, when enabled. It provides a graphical representation of the number of synchronizations in each phase. The same horizontal 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 clicking View » Go To.

  See “Utilization Graph pane” on page 183.

- **Chart**  
  Chart is the third pane and is always displayed. It provides a graphical representation of synchronizations, colored by synchronization phase. 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 clicking View » Go To.

  See “Chart pane” on page 184.

- **Overview**  
  Overview is the bottom pane, when enabled. It shows an overview of all synchronizations in the session. This pane contains a box outline called the Marquee Tool that indicates and can select the region appearing in the Chart and Utilization Graph panes.

  See “Overview pane” on page 185.

In addition, there is an Options window that you can use to customize the display and property sheets that can be displayed. See “Options window” on page 186.
**Details Table pane**

The **Details Table** provides information about the synchronizations, including phase times. All times are measured by the MobiLink server. Some phase 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 clicking **Tools » Options** and then opening the **Table** tab. For information about the statistics that are available, see “MobiLink synchronization statistical properties” on page 191.

The following columns appear by default:

- **number** Identifies each synchronization. This number is assigned by the MobiLink server, not by the MobiLink Profiler, so it does not necessarily start at 1 in any given MobiLink Profiler session and is not necessarily received in numerical order. This number is the same as the synchronization number shown in MobiLink server warnings, errors, and logs. You can see the same number in the **Synchronization Properties** window. See “Synchronization properties” on page 187.

- **remote_id** The unique identifier of the remote database. See “Remote IDs” [*MobiLink - Client Administration*].

- **user** The MobiLink synchronization user. See “MobiLink users” [*MobiLink - Client Administration*].

- **version** The version of the synchronization script.

  See “Script versions” on page 232.

- **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.

All the following phase times are in seconds.

- **sync_request phase** 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`, in which case 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`.

- **receive_upload phase** The time taken from the first bytes of the upload stream being received by the MobiLink server until the upload stream from the remote database has been completely received. The upload stream includes table definitions and the remote database rows being uploaded, so 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 phase** The time required to acquire a free database worker thread.

- **connect phase** 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 phase  The time for MobiLink to validate the synchronization request, the user name, and 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 phase  The time to run your begin_synchronization script, if one was run.

apply_upload phase  The time 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 phase  The time to run your prepare_for_download script, if one was run.

fetch_download phase  The time to fetch the rows to be downloaded from the consolidated database. This is the time between the begin_download script and the end_download script.

download phase  The time to run the end_synchronization script, if one was run.

send_download phase  The time taken to send the download stream to the remote database.

wait_for_download_ack phase  If download acknowledgement is enabled, this includes the time spent waiting for the download to be applied to the remote database and for the remote database to send the download acknowledgement.

generate_db_worker_for_download_ack phase  If download acknowledgement is enabled, this includes the time spent waiting for a free database worker thread after the download acknowledgement was received.

connect_for_download_ack phase  If download acknowledgement is enabled, this includes the time required by the database worker thread to make a database connection if a new database connection is needed.

nonblocking_download_ack phase  If download acknowledgement is enabled, this includes the time required for the publication_nonblocking_download_ack connection and nonblocking_download_ack connection events.

To sort the table by a specific column, click the column heading. If new data appears in the MobiLink Profiler, 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 the number of synchronizations in each phase in a time graph.

For more information about the data available in this pane, see “How the Utilization Graph works” on page 184.

The Utilization Graph uses the same horizontal scrollbar, horizontal time labels, and horizontal zoom level as the Chart. An instant in time lines up vertically between the Graph pane and the Chart pane.
There are two ways to select the time range that is displayed in the **Graph** and **Chart**:

- From the **View** menu, click **Go To**.
- In the **Overview** pane, move the **Marquee Tool**. The **Marquee Tool** is the small box that appears in the **Overview** pane.

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.

Drag your mouse in the **Utilization Graph** pane to see data for a range of samples. The **Sample Range Properties** window appears.

### See also
- “How the Utilization Graph works” on page 184
- “Marquee tool” on page 186
- “Sample properties” on page 187

### How the Utilization Graph works
To customize the **Utilization Graph**, click **Tools** » **Options** and click the **Graph** tab. This tab identifies the **Utilization Graph** times by color, and allows you to customize the graph.

#### Phase counters
Each property shows the number of synchronizations currently in that phase.

#### 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 default columns in 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 187.

#### Grouping data by remote ID or compactly
To group the data by remote ID, click **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. Click **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 and Utilization Graph:

- **Zoom options**  
  Zoom options in the View menu and zoom buttons on the toolbar 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** 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** 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, drag to change the Marquee Tool. The Marquee Tool is the box that appears in the Overview pane. See “Marquee tool” on page 186.

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 186.

Overview pane

The Overview pane shows an overview of the entire MobiLink Profiler session. You can navigate through the session using the Marquee Tool, which is the box inside the Overview pane.

By default, active synchronizations, blocked synchronizations, completed synchronizations, and failed synchronizations are represented with colors via watches. To set the colors, open the MobiLink Profiler, click Tools » Options, and then click the Overview tab or edit the corresponding watches by clicking Tools » Watch Manager and then Edit.

See:

- “Options window” on page 186
- “Statistic customization” on page 188

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 Profiler window. In the **Options** window, click 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.

### Changing the color of the Marquee Tool

You can change the color of the **Marquee Tool**.

**Prerequisites**

There are no prerequisites for this task.

**Change the color of the Marquee Tool**

1. Click **Tools » Options**.
2. Click the **Overview** tab.
3. Select a new color in the **Marquee** field.
4. Click **OK**.

**Results**

The color of the **Marquee Tool** is changed.

### Options window

Options allow you to specify many settings, including colors and patterns for the graphical display in the **Chart** pane, **Utilization Graph** pane, and the **Overview** pane.

From the **Options** window: **General** tab, you can change the profiling database used to store the profiling data and you can recreate the profiling database, which removes all previous profiling sessions.

To open the **Options** window, open the MobiLink Profiler and click **Tools » Options**.
Session properties

The Session Properties window provides statistics about the profiling session. It provides property values for the current profiling session. To open the Session Properties window, click File » Properties.

Sample properties

To open the Sample Properties window, click in the Graph pane for the time period that you want to examine. The Sample Properties window provides detailed statistics for time intervals. Each time interval is about one second long. Samples are numbered by the MobiLink Profiler to reflect the order in which they were received.

The Sample Properties window has three tabs:

- **General** Provides a high-level breakdown of what your synchronizations were doing at the time the sample was taken.

- **Phases** Provides counts of the phases your synchronizations were in at the time the sample was taken.

- **Events** Provides information about event scripts being run during synchronization.

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 phase, it is identified as Hidden in the Phases tab of the Sample Properties window; otherwise, the color is shown.

The Sample Range Properties window shows information for the multiple samples if you selected multiple samples by dragging in the Utilization Graph.

The Sample Range Properties window has the same tabs as the Sample Properties window. However, average and maximum values are displayed for the range.

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.

The Synchronization page shows warning and/or errors, the Events page shows how often the event scripts were called and how long they took. If the MobiLink server detected that the synchronization was blocked, then a Blocked page is available.

For descriptions of the quantities displayed on any page of the Synchronization Properties window, click Help.
For an explanation of the statistics in the Synchronization Properties window, see “MobiLink synchronization statistical properties” on page 191.

Statistic customization

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 Profiler 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 four predefined watches (Active, Blocked, 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 or Overview panes.

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 Profiler connects to the same MobiLink server, it shows up as a short synchronization in any MobiLink Profilers that are already connected. The MobiLink Profiler synchronization has the version name for_ML_Monitor_only. You can hide this MobiLink Profiler synchronization by only enabling watches that have the following condition:

- Property Set to Version
- Operator Set to is not equal to
- Value Set to for_ML_Monitor_only

Creating a new watch

Add a watch to display synchronizations that meet the defined watch criteria.

Prerequisites

There are no prerequisites for this task.
Task
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 synchronization statistical properties” on page 191.
4. Click Add. (You must click Add to save the condition.)
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.
8. Click OK.

Results
The new watch is created.

Using the profiling database
In Sybase Central, you can use predefined views to review and analyze data in the profiling database.

Prerequisites
You must have run at least one profiling session.
You must know the name and location of the profiling database. To determine this information, go to the General page of the Options window.

Context and remarks
This task assumes you are using the default profiling database.
You can also use Interactive SQL to work with Profiler views.

Task
1. Use the SQL Anywhere 16 plug-in to connect to the profiling database, using the following options:
   - User ID Type mlprofiler for the User ID.
   - Password Type sql for the password.
   - Action Choose Start and connect to a database on this computer.
● **Database file**  Enter the path information for the profiling database or click **Browse** to select the file. The default database file is _mlprofiler.db_ in a folder called _MLProfiler16_ in your Documents folder.

● **Server name**  Type _MLProfilerDB_.

● **Start line**  To set the initial memory for caching database pages and other database server information, type the following:

```
  dbeng16.exe -c 1g
```

2. Click **Connect**.

3. Expand the _mlprofiler_ database and double-click **Views** to see a list of the MobiLink Profiler views.

4. Select a view. The following views are available:

   ● **category_samples** (base view for category sampling data)
   ● **data_event_statistics**
   ● **data_event_times**
   ● **data_phase_statistics**
   ● **data_phase_times**
   ● **event_samples** (base view for event sampling data)
   ● **event_statistics**
   ● **event_times**
   ● **event_total_times**
   ● **phase_samples** (base view for phase sampling data)
   ● **phase_statistics**
   ● **phase_times**
   ● **server_cumulative_samples** (base view for cumulative server-related sampling data)
   ● **server_snapshot_samples** (base view for non-cumulative server-related sampling data)
   ● **sync_as_csv** (view like the old MobiLink Monitor .csv file format)
   ● **sync_blocked**
   ● **sync_statistics**
   ● **sync_times**
   ● **syncs** (base view for synchronizations)

The SQL pane on the right includes a comment at the top that describes the selected view.

```
Note
The server-related sampling data is for metrics that are also available in the SQL Anywhere Monitor for MobiLink.
```

**Results**

Data from the profiling database is displayed for each view in the **Data** page.
Next

- Review the profiling data.
- Use these views in your queries.

Example

The following sample query shows the event scripts that consumed the most time for all synchronizations in the second session:

```sql
select * from event_total_times where session_id = 2 order by 1 desc
```

The following sample query shows the fastest synchronization completion rates for all the sessions:

```sql
select
    max( "Successful syncs/s" ) as "Max syncs/s",
    session_id
from server_throughput_samples
group by session_id
order by 1 desc, 2
```

MobiLink synchronization statistical properties

The following is a list of the statistical properties for synchronizations that are available in the MobiLink Profiler. 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 “Statistic customization” on page 188.

For more information about the Details Table, see “Details Table pane” on page 182.

For more information about the Synchronization Properties window, see “Synchronization properties” on page 187.

Synchronization statistics

MobiLink statistical properties return the following information for synchronizations:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>active</td>
<td>True if the synchronization is in progress.</td>
</tr>
<tr>
<td>apply_upload phase</td>
<td>Time required for the uploaded data to be applied to the consolidated database.</td>
</tr>
<tr>
<td>authenticate_user phase</td>
<td>Total time to perform user authentication, including executing the authenticate_* events.</td>
</tr>
<tr>
<td>begin_sync phase</td>
<td>Total time for the begin_synchronization event.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>client</td>
<td>The type of MobiLink client and full client version. For example, Dbmlsync 16.0.0.xxxx.</td>
</tr>
<tr>
<td>completed</td>
<td>True if the synchronization completed successfully.</td>
</tr>
<tr>
<td>conflicted_updates</td>
<td>Number of update rows that caused conflict. A row is included only when a resolve conflict script was successfully called for it.</td>
</tr>
<tr>
<td>connect phase</td>
<td>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.</td>
</tr>
<tr>
<td>connect_for_download_ack_phase</td>
<td>Time required by the database worker thread to make a database connection if a new database connection is needed for a download acknowledgement.</td>
</tr>
<tr>
<td>connection_retries</td>
<td>Number of times the MobiLink server retried the connection to the consolidated database.</td>
</tr>
<tr>
<td>download</td>
<td>This property indicates the synchronization included a download command.</td>
</tr>
<tr>
<td>download_ack</td>
<td>Can be none or non-blocking.</td>
</tr>
<tr>
<td>download_bytes</td>
<td>Amount of memory used within the MobiLink server to store the download and send to the remote database (before any encryption or compression).</td>
</tr>
<tr>
<td>download_deleted_rows</td>
<td>Number of row deletions fetched from the consolidated database by the MobiLink server (using download_delete_cursor scripts).</td>
</tr>
<tr>
<td>download_errors</td>
<td>Number of errors that occurred during the download.</td>
</tr>
<tr>
<td>download_fetched_rows</td>
<td>Number of rows fetched from the consolidated database by the MobiLink server (using download_cursor scripts).</td>
</tr>
<tr>
<td>download_filtered_rows</td>
<td>Number of fetched rows that were not downloaded to the MobiLink client because they matched rows that the client uploaded.</td>
</tr>
<tr>
<td>download_warnings</td>
<td>Number of warnings that occurred during the download.</td>
</tr>
<tr>
<td>duration</td>
<td>Total time for the synchronization, as measured by the MobiLink server.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>end_sync phase</td>
<td>Total time for the end_synchronization event.</td>
</tr>
<tr>
<td>fetch_download phase</td>
<td>Time required to fetch the rows to be downloaded from the consolidated database to create the download stream.</td>
</tr>
<tr>
<td>get_db_worker phase</td>
<td>Time required to acquire a free database worker thread.</td>
</tr>
<tr>
<td>get_db_worker_for_download_ack</td>
<td>Time spent waiting for a free database worker thread after the download acknowledgement has been received.</td>
</tr>
<tr>
<td>has_blocked</td>
<td>True if blocking is detected by the MobiLink server.</td>
</tr>
<tr>
<td>ignored_deletes</td>
<td>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.</td>
</tr>
<tr>
<td>ignored_inserts</td>
<td>Total number of upload insert rows that were ignored. They were ignored because 1) there is no upload_insert script in normal 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.</td>
</tr>
<tr>
<td>ignored_updates</td>
<td>Number of upload update rows that caused conflict but a resolve conflict script was not successfully called or no upload_update script was defined.</td>
</tr>
<tr>
<td>nonblocking_download_ack phase</td>
<td>Time required for the publication_nonblocking_download_ack connection and nonblocking_download_ack connection events.</td>
</tr>
<tr>
<td>number</td>
<td>Synchronization number.</td>
</tr>
<tr>
<td>prepare_for_download phase</td>
<td>Total time for the prepare_for_download event.</td>
</tr>
<tr>
<td>receive_upload phase</td>
<td>Phase time for receiving the upload.</td>
</tr>
<tr>
<td>remote_id</td>
<td>Remote ID that uniquely identifies the remote database.</td>
</tr>
<tr>
<td>send_download phase</td>
<td>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.</td>
</tr>
<tr>
<td>server</td>
<td>The MobiLink server name or host:port.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>start_time</td>
<td>Date-time (in ISO-8601 extended format) for the start of the synchronization.</td>
</tr>
<tr>
<td>sync_deadlocks</td>
<td>Number of deadlocks in the consolidated database that were detected for the synchronization.</td>
</tr>
<tr>
<td>sync_errors</td>
<td>Total number of errors that occurred for the synchronization.</td>
</tr>
<tr>
<td>sync_request_phase</td>
<td>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.</td>
</tr>
<tr>
<td>sync_tables</td>
<td>Number of client tables that were involved in the synchronization.</td>
</tr>
<tr>
<td>sync_warnings</td>
<td>Number of warnings that occurred for the synchronization.</td>
</tr>
<tr>
<td>upload</td>
<td>Indicates the synchronization included an upload command.</td>
</tr>
<tr>
<td>upload_bytes</td>
<td>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.</td>
</tr>
<tr>
<td>upload_deadlocks</td>
<td>Number of deadlocks in the consolidated database that were detected during the upload.</td>
</tr>
<tr>
<td>upload_deleted_rows</td>
<td>Number of rows that were successfully deleted from the consolidated database.</td>
</tr>
<tr>
<td>upload_errors</td>
<td>Number of errors that occurred during the upload.</td>
</tr>
<tr>
<td>upload_inserted_rows</td>
<td>Number of rows that were successfully inserted in the consolidated database.</td>
</tr>
<tr>
<td>upload_updated_rows</td>
<td>Number of rows that were successfully updated in the consolidated database.</td>
</tr>
<tr>
<td>upload_warnings</td>
<td>Number of warnings that occurred during the upload.</td>
</tr>
<tr>
<td>user</td>
<td>MobiLink user name.</td>
</tr>
<tr>
<td>version</td>
<td>Name of the synchronization version.</td>
</tr>
<tr>
<td>wait_for_download_ack</td>
<td>Time spent waiting for the download to be applied to the remote database and for the remote database to send the download acknowledgement.</td>
</tr>
</tbody>
</table>
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.
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.

See also
- “Download-only publications” [MobiLink - Client Administration]

File-based download setup

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.

1. Create a file-definition database.
   See “File-definition database” on page 196.

2. At the consolidated database, create scripts with a new script version.
   See “Changes at the consolidated database” on page 197.

3. Create a download file.
   See “Download file creation” on page 197.

4. Apply the download file.
   See “Synchronization of new remotes” on page 198.

Other resources for getting started
- “File-based download examples”

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 recommended 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 database is unable to apply a download file will be proportional to the frequency with which the remote database performs normal (connected) synchronizations and the amount of concurrent activity in the consolidated database.

See “-dsd mlsrv16 option” on page 54.

See also

- “Script versions” on page 232
- “begin_publication connection event” on page 283
- “modify_last_download_timestamp connection event” on page 360

Download file creation

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.

- **-bg option** Use the -bg option to create a download file that can be used by remotes that have never synchronized.

**See also**
- “-be dbmlsync option” [MobiLink - Client Administration]
- “sp_hook_dbmlsync_validate_download_file” [MobiLink - Client Administration]
- “-bg dbmlsync option” [MobiLink - Client Administration]

### Synchronization of new remotes

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 database 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 database 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 also
- “MobiLink generation numbers” on page 201
- “-bg dbmlsync option” [MobiLink - Client Administration]

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 database did not apply previous file-based downloads.

See “Automatic validation” on page 200.
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 201.

 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 202.

**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 database never misses operations that occur on the consolidated database. Usually when a file-based download fails based on this check, the remote database 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 database's last download timestamp. The remote database's last download timestamp is the time (at the consolidated database) up to which the remote database has received all changes that are to be downloaded. The remote database's last download time is updated each time the remote database 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 database 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 database 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 283
● “end_publication connection event” on page 321
● “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.

See also

- “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 -dba DBA,sql scons.db
dbinit -dba DBA,sql sremote.db
dbinit -dba DBA,sql 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.

```
dbeng16 -n scons_eng scons.db
dbeng16 -n scons_eng scons.db
dbeng16 -n sremote_eng sremote.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 16\MobiLink\setup\syncsa.sql"
```

Start the MobiLink server:
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:

```sql
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.

```sql
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.

```sql
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', 'download_cursor', 'SELECT pk, c1 FROM T1' );
CALL ml_add_table_script( 'normal', 'T1', 'download_delete_cursor', '--{ml_ignore}' );
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:

```sql
CREATE PROCEDURE begin_pub ( INOUT generation_num integer, IN username varchar(128), IN pubkey 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} ) });
```

The following command creates the consolidated database example snapshot:

```sql
start mlsrv16 -v+ -c "DSN=fbd_demo" -zu+ -ot scons.txt
```
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.

```sql
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:

```sql
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.

```sql
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:

```sql
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:

```sql
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.

```sql
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:

```bash
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 database has the data:

```sql
SELECT * FROM T1
```

**Clean up the snapshot example**

The following commands stop all three database servers and erase the files.

```bash
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.

```bash
dbind -dba DBA,sql tcons.db
dbind -dba DBA,sql tremote.db
dbind -dba DBA,sql 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.

```bash
dbeng16 -n tfdef_eng tfdef.db
dbeng16 -n tcons_eng tcons.db
dbeng16 -n tremote_eng tremote.db
dbsn -y -w tfbd_demo -c "SERVER=tcons_eng;DBF=tcons.db;UID=DBA;PWD=sql;START=off;ASTOP=off"
```

Open Interactive SQL, connect to `tcons.db` and run the MobiLink setup script. For example:

```sql
read "C:\Program Files\SQL Anywhere 16\MobiLink\setup\syncsa.sql"
```

Start the MobiLink server:

```bash
start mlsrv16 -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:

```sql
CREATE TABLE T1 (
    pk  INTEGER PRIMARY KEY,
    c1  INTEGER,
    last_modified TIMESTAMP DEFAULT CURRENT_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.

```sql
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.

```sql
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.

```sql
CALL ml_add_connection_script( 'filebased',
    'begin_publication',
    '{ call begin_pub( {ml s.generation_number},
        {ml s.username},
```
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 database that has missed all the download files created in the last five days has to perform a normal synchronization before being able to apply any more file-based downloads.

```sql
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.

```sql
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.

```sql
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
```
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:

```sql
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.

```bash
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.

```bash
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 database has the data:

```sql
SELECT * FROM T1
```

Clean up the timestamp example

The following commands stop all three database servers and then erase the files.

```bash
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.

Synchronization script writing

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 information about every script you can write, see “Synchronization events” on page 248.
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 “Synchronization scripts in .NET” on page 497.

For information about writing scripts in Java, see “Synchronization script writing in Java” on page 427.

For information about how to implement synchronization scripts, see “Synchronization techniques” on page 98.

**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. This simple example does not use download deletes so the script is defined 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 248.

For the details of upload processing, see “Scripts to upload rows” on page 239.

For the details of download processing, see “Scripts to download rows” on page 242.

**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 248.
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 with other scripts to implement your synchronization business logic.

- **table-level scripts**  These scripts perform actions specific to one synchronization and one particular remote table. These scripts are used with other scripts to implement your synchronization business logic, including conflict resolution.

See also

- “Connection scripts” on page 215
- “Table scripts” on page 215

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 also

- “ml_global script version” on page 233

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 248.

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 230.

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:

<table>
<thead>
<tr>
<th>Type of named parameter</th>
<th>Prefix</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>System parameters.</td>
<td>s.</td>
<td>{ml s.remote_id}</td>
</tr>
<tr>
<td>Row parameters. (The column name. If the column contains spaces, enclose it in double quotes or square brackets.)</td>
<td>r.</td>
<td>{ml r.cust_id}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{ml r.&quot;Column name&quot;}</td>
</tr>
<tr>
<td>Type of named parameter</td>
<td>Prefix</td>
<td>Examples</td>
</tr>
<tr>
<td>----------------------------------------------------------------------------------------</td>
<td>--------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>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.)</td>
<td>o.</td>
<td>{ml o.cust_name}</td>
</tr>
<tr>
<td></td>
<td></td>
<td>{ml o.&quot;Column name&quot;}</td>
</tr>
<tr>
<td>Authentication parameters. See “Authentication parameters” on page 231.</td>
<td>a.</td>
<td>{ml a.1}</td>
</tr>
<tr>
<td>User-defined parameters. See “User-defined named parameters” on page 230.</td>
<td>u. or ui.</td>
<td>● {ml u.varname} Use if the parameter if the parameter will be updated. User-defined parameters prefixed with u. are in/out.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● {ml ui.varname} Use if the parameter will only be referenced. User-defined parameters prefixed with ui. are input-only.</td>
</tr>
</tbody>
</table>

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 ) }
```

**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 216 and “User-defined named parameters” on page 230.

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.

Some parameters are optional. A parameter is optional only if no subsequent parameters are specified. For example, you must use parameter 1 if you are going to use parameter 2. The parameters must be in the order specified for each event. See “Synchronization events” on page 248.

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

<table>
<thead>
<tr>
<th>System parameter</th>
<th>Event type</th>
<th>Event parameter is available in</th>
</tr>
</thead>
</table>
| action_code          | connection | The action_code parameter is available in the following events:  
|                      |            | ● handle_error  
|                      |            | ● handle_odbc_error  
|                      |            | ● report_error  
|                      |            | ● report_odbc_error  
| authentication_message | connection | The authentication_message parameter is available in the following events:  
|                      |            | ● authenticate_parameters  
|                      |            | ● authenticate_user  
|                      |            | ● authenticate_user_hashed  
| authentication_status | connection | The authentication_status parameter is available in the following events:  
|                      |            | ● authenticate_parameters  
|                      |            | ● authenticate_user  
<p>|                      |            | ● authenticate_user_hashed  |</p>
<table>
<thead>
<tr>
<th>System parameter</th>
<th>Event type</th>
<th>Event parameter is available in</th>
</tr>
</thead>
</table>
| 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:  
<p>|                  |                         | • handle_error                   । modify_error_message । report_error |</p>
<table>
<thead>
<tr>
<th>System parameter</th>
<th>Event type</th>
<th>Event parameter is available in</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_message</td>
<td>connection</td>
<td>The error_message parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• handle_error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• handle_odbc_error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• modify_error_message</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• report_error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• report_odbc_error</td>
</tr>
<tr>
<td>errors</td>
<td>connection and table</td>
<td>The errors parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• download_statistics</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• synchronization_statistics</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• upload_statistics</td>
</tr>
<tr>
<td>event_name</td>
<td>connection and table</td>
<td>The event_name parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• time_statistics</td>
</tr>
<tr>
<td>fetched_rows</td>
<td>connection and table</td>
<td>The fetched_rows parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• download_statistics</td>
</tr>
<tr>
<td>file_authentication_code</td>
<td>connection</td>
<td>The file_authentication_code parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• authenticate_file_transfer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• authenticate_file_upload</td>
</tr>
<tr>
<td>filename</td>
<td>connection</td>
<td>The filename parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• authenticate_file_transfer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• authenticate_file_upload</td>
</tr>
<tr>
<td>file_size</td>
<td>connection</td>
<td>The file_size parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• authenticate_file_upload</td>
</tr>
<tr>
<td>filtered_rows</td>
<td>connection and table</td>
<td>The filtered_rows parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• download_statistics</td>
</tr>
<tr>
<td>System parameter</td>
<td>Event type</td>
<td>Event parameter is available in</td>
</tr>
<tr>
<td>--------------------------</td>
<td>--------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>generation_number</td>
<td>connection</td>
<td>The filtered_rows parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• begin_publication</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• end_publication</td>
</tr>
<tr>
<td>hashed_new_password</td>
<td>connection</td>
<td>The hashed_new_password parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• authenticate_user_hashed</td>
</tr>
<tr>
<td>hashed_password</td>
<td>connection</td>
<td>The hashed_password parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• authenticate_user_hashed</td>
</tr>
<tr>
<td>ignored_deletes</td>
<td>connection and table</td>
<td>The ignored_deletes parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• upload_statistics</td>
</tr>
<tr>
<td>ignored_inserts</td>
<td>connection and table</td>
<td>The ignored_inserts parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• upload_statistics</td>
</tr>
<tr>
<td>ignored_updates</td>
<td>connection and table</td>
<td>The ignored_updates parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• upload_statistics</td>
</tr>
<tr>
<td>inserted_rows</td>
<td>connection and table</td>
<td>The inserted_rows parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• upload_statistics</td>
</tr>
<tr>
<td>last_download</td>
<td>connection</td>
<td>The last_download parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• begin_download</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• end_download</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• modify_last_download_timestamp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• modify_next_last_download_timestamp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• nonblocking_download_ack</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• prepare_for_download</td>
</tr>
<tr>
<td>System parameter</td>
<td>Event type</td>
<td>Event parameter is available in</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>last_publication_download</td>
<td>connection</td>
<td>The last_publication_download parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● begin_publication</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● end_publication</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● publication_nonblocking_download_ack</td>
</tr>
<tr>
<td>last_publication_upload</td>
<td>connection</td>
<td>The last_publication_upload parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● begin_publication</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● end_publication</td>
</tr>
<tr>
<td>last_table_download</td>
<td>table</td>
<td>The last_table_download parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● begin_download</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● begin_download_deletes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● begin_download_rows</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● download_cursor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● download_delete_cursor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● end_download</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● end_download_deletes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● end_download_rows</td>
</tr>
<tr>
<td>maximum_time</td>
<td>connection and table</td>
<td>The maximum_time parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● time_statistics</td>
</tr>
<tr>
<td>minimum_time</td>
<td>connection and table</td>
<td>The minimum_time parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● time_statistics</td>
</tr>
<tr>
<td>new_password</td>
<td>connection</td>
<td>The new_password parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● authenticate_user</td>
</tr>
<tr>
<td>next_last_download</td>
<td>connection</td>
<td>The next_last_download parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● generate_next_last_download_timestamp</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● modify_next_last_download_timestamp</td>
</tr>
<tr>
<td>System parameter</td>
<td>Event type</td>
<td>Event parameter is available in</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>number_of_calls</td>
<td>connection and table</td>
<td>The number_of_calls parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• time_statistics</td>
</tr>
<tr>
<td>odbc_state</td>
<td>connection</td>
<td>The odbc_state parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• handle_odbc_error</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• report_odbc_error</td>
</tr>
<tr>
<td>password</td>
<td>connection</td>
<td>The password parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• authenticate_user</td>
</tr>
<tr>
<td>publication_name</td>
<td>connection</td>
<td>The publication_name parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• begin_publication</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• end_publication</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• publication_nonblocking_download_ack</td>
</tr>
<tr>
<td>System parameter</td>
<td>Event type</td>
<td>Event parameter is available in</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>remote_id</td>
<td>connection and table</td>
<td>The remote_id parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- authenticate_parameters (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- authenticate_user (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- authenticate_user_hashed (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- begin_download (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- begin_download_deletes (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- begin_download_rows (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- begin_publication (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- begin_synchronization (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- begin_upload (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- begin_upload_deletes (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- begin_publish_rows (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- download_cursor (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- download_delete_cursor (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- download_statistics (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- end_download (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- end_download_deletes (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- end_download_rows (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- end_publication (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- end_synchronization (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- end_upload (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- end_upload_deletes (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- end_upload_rows (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- generate_next_last_download_timestamp (connection)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- handle_error (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- handle_odbc_error (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- modify_error_message (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- modify_last_download_timestamp (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- modify_next_last_download_timestamp (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- modify_user (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- nonblocking_download_ack (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- prepare_for_download (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- publication_nonblocking_download_ack (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- report_error (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- report_odbc_error (connection event)</td>
</tr>
<tr>
<td>System parameter</td>
<td>Event type</td>
<td>Event parameter is available in</td>
</tr>
<tr>
<td>------------------</td>
<td>------------</td>
<td>---------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● resolve_conflict (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● synchronization_statistics (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● time_statistics (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_delete (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_fetch (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_fetch_column_conflict (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_insert (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_new_row_insert (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_old_row_insert (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_statistics (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_update (table event)</td>
</tr>
</tbody>
</table>

remote_key | connection | The remote_key parameter is available in the following events: |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>● authenticate_file_transfer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● authenticate_file_upload</td>
</tr>
<tr>
<td>System parameter</td>
<td>Event type</td>
<td>Event parameter is available in</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>script_version</td>
<td>connection and table</td>
<td>The script_version parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● authenticate_file_transfer (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● authenticate_file_upload (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● authenticate_parameters (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● authenticate_user (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● authenticate_user_hashed (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● begin_download (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● begin_download_deletes (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● begin_download_rows (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● begin_publication (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● begin_synchronization (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● begin_upload (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● begin_upload_deletes (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● begin_upload_rows (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● download_cursor (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● download_delete_cursor (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● download_statistics (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● end_download (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● end_download_deletes (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● end_download_rows (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● end_publication (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● end_synchronization (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● end_upload (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● end_upload_deletes (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● end_upload_rows (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● generate_next_last_download_timestamp (connection)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● handle_DownloadData (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● handle_error (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● handle_odbc_error (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● modify_error_message (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● modify_last_download_timestamp (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● modify_next_last_download_timestamp (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● modify_user (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● nonblocking_download_ack (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● prepare_for_download (connection event)</td>
</tr>
<tr>
<td>System parameter</td>
<td>Event type</td>
<td>Event parameter is available in</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● publication_nonblocking_download_ack (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● report_error (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● report_odbc_error (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● resolve_conflict (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● synchronization_statistics (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● time_statistics (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_delete (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_fetch (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_fetch_column_conflict (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_insert (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_new_row_insert (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_old_row_insert (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_statistics (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_update (table event)</td>
</tr>
</tbody>
</table>

**Note**


subdir | connection | The subdir parameter is available in the following events:  
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>● authenticate_file_transfer</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● authenticate_file_upload</td>
</tr>
</tbody>
</table>

subscription_id | connection | The subscription_id parameter is available in the following events:  
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>● begin_publication</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● end_publication</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● publication_nonblocking_download_ack</td>
</tr>
</tbody>
</table>

synchronization_ok | connection and table | The synchronization_ok parameter is available in the following events:  
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>● end_synchronization</td>
</tr>
<tr>
<td>System parameter</td>
<td>Event type</td>
<td>Event parameter is available in</td>
</tr>
<tr>
<td>--------------------------</td>
<td>----------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>synchronized_tables</td>
<td>connection</td>
<td>The synchronized_tables parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· synchronization_statistics</td>
</tr>
<tr>
<td>table</td>
<td>connection and table</td>
<td>The table parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· begin_download (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· begin_download_deletes (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· begin_download_rows (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· begin_synchronization (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· begin_upload (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· begin_upload_deletes (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· begin_upload_rows (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· download_statistics (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· end_download (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· end_download_deletes (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· end_download_rows (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· end_synchronization (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· end_upload (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· end_upload_deletes (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· end_upload_rows (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· handle_error (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· handle_odbc_error (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· report_error (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· report_odbc_error (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· resolve_conflict (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· synchronization_statistics (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· time_statistics (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· upload_statistics (table event)</td>
</tr>
<tr>
<td>total_time</td>
<td>connection and table</td>
<td>The total_time parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· time_statistics</td>
</tr>
<tr>
<td>updated_rows</td>
<td>connection and table</td>
<td>The updated_rows parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>· upload_statistics</td>
</tr>
<tr>
<td>System parameter</td>
<td>Event type</td>
<td>Event parameter is available in</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------</td>
<td>---------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>username</td>
<td>connection and table</td>
<td>The username parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• authenticate_file_transfer (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• authenticate_file_upload (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• authenticate_parameters (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• authenticate_user (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• authenticate_user_hashed (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• begin_download (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• begin_download_deletes (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• begin_download_rows (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• begin_publication (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• begin_synchronization (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• begin_upload (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• begin_upload_deletes (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• begin_upload_rows (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• download_cursor (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• download_delete_cursor (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• download_statistics (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• end_download (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• end_download_deletes (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• end_download_rows (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• end_publication (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• end_synchronization (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• end_upload (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• end_upload_deletes (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• end_upload_rows (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• generate_next_last_download_timestamp (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• handle_error (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• handle_odbc_error (connection error)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• modify_error_message (connection error)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• modify_last_download_timestamp (connection error)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• modify_next_last_download_timestamp (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• modify_user (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• nonblocking_download_ack (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• prepare_for_download (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• publication_nonblocking_download_ack (connection event)</td>
</tr>
<tr>
<td>System parameter</td>
<td>Event type</td>
<td>Event parameter is available in</td>
</tr>
<tr>
<td>------------------</td>
<td>------------</td>
<td>----------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● report_error (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● report_odbc_error (connection event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● resolve_conflict (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● synchronization_statistics (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● time_statistics (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_delete (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_fetch (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_fetch_column_conflict (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_insert (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_new_row_insert (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_old_row_insert (table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_statistics (connection and table event)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_update (table event)</td>
</tr>
<tr>
<td>warnings</td>
<td>connection and table</td>
<td>The warnings parameter is available in the following events:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● download_statistics</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● synchronization_statistics</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● upload_statistics</td>
</tr>
</tbody>
</table>

**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:

```sql
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 table script references var1, whose value is custom_value:

CALL ml_add_table_script (
    'version1',
    'MyTable',
    'download_cursor',
    '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 can be specified using named parameters. If named parameters are used, the authentication parameters must be prefaced with the letter a, such as {ml a.1}. The parameters must be numbers starting at 1, with a limit of 255. Each parameter can be a maximum of 4000 bytes. 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 216
- 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:

```c
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:

```bash
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:

```sql
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 to a consolidated database” on page 234.

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 621.

### 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: `mlsis_1` or `mlqa_1`. These names are used internally by MobiLink.

### 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 “Script versions and 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 to a consolidated database

A script version identifies a set of scripts. When working in Sybase Central, you must add a script version name to your consolidated database before you can add any connection scripts.

**Prerequisites**

There are no prerequisites for this task.

**Context and remarks**

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.

To be able to perform schema changes without synchronizing, you must add a script version to the synchronization subscription using SQL syntax.

You can add a script version in the same operation as adding a connection script or table script using system procedures. See “System procedures to add or delete scripts” on page 606.

See “Script versions” on page 232.

**Task**

1. From the View menu, click 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. Click the Versions folder and click File » New » Version.

4. Follow the instructions in the Create Script Version Wizard.

**Results**

The script version is created.

Removing a script version from a consolidated database

Use the following procedure to remove a script version and its associated scripts from the consolidated database.

**Prerequisites**

There are no prerequisites for this task.
**Task**

1. From the View menu, click **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.

**Results**

The script version and its associated scripts are removed from the consolidated database.

**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. 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 “Ignored scripts” on page 239.

**Script additions and deletions**

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 to a consolidated database” on page 234.

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 information about the stored procedures that you can use to add or delete scripts, see:

- “ml_add_connection_script system procedure” on page 609
- “ml_add_table_script system procedure” on page 624
- “ml_add_dnet_connection_script system procedure” on page 610
- “ml_add_dnet_table_script system procedure” on page 611
- “ml_add_java_connection_script system procedure” on page 612
- “ml_add_java_table_script system procedure” on page 614

Adding a connection script

Use the following procedure to add a connection script using Sybase Central.

Prerequisites

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 to a consolidated database” on page 234.

Task

1. From the View menu, click 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. Click Connection Scripts and click New » Connection Script.
4. Follow the instructions in the Create Connection Script Wizard.

Results

The connection script is created.
Deleting a connection script

Use the following procedure to delete a connection script using Sybase Central.

Prerequisites

There are no prerequisites for this task.

Task

1. From the View menu, click 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 click Delete.

5. Click Yes.

Results

The connection script is deleted.

Adding a table script

Use the following procedure to add a table script using Sybase Central.

Prerequisites

There are no prerequisites for this task.

Task

1. From the View menu, click 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 click New » Table Script.

5. Follow the instructions in the Create Table Script Wizard.

Results

The table script is created.
Deleting a table script

Use the following procedure to delete a table script.

Prerequisites

There are no prerequisites for this task.

Task

1. From the View menu, click 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 click Delete.
6. Click Yes.

Results

The table script is deleted.

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 information about 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 %SQLANY16%\MobiLink\Setup and are called:

<table>
<thead>
<tr>
<th>Consolidated database</th>
<th>Setup file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptive Server Enterprise</td>
<td>syncase.sql</td>
</tr>
<tr>
<td>IBM DB2 LUW</td>
<td>syncdb2.sql</td>
</tr>
<tr>
<td>Microsoft SQL Server</td>
<td>syncmss.sql</td>
</tr>
</tbody>
</table>
Ignored 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.

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 information about the upload process, see “Events during upload” on page 255.

For techniques for uploading rows using Java or .NET synchronization logic, see “Row uploads and downloads” on page 508.

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].

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

- “Scripts to upload rows” on page 239
- “upload_insert table event” on page 407

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

- “Scripts to upload rows” on page 239
- “upload_update table event” on page 422

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.

```sql
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

- “Scripts to upload rows” on page 239
- “upload_delete table event” on page 401

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:

```sql
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:

```sql
SELECT pk1, pk2, val  
FROM uf_example  
WHERE pk1 = {ml r.pk1} and pk2 = {ml r.pk2}
```

See “upload_fetch table event” on page 403.

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.
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 “Result sets from stored procedure calls” on page 127.

The MobiLink server downloads data in a single transaction. For information about the download process, see “Events during download” on page 257.

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 “Last download times in scripts” on page 100.

- 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.

  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 369 and “publication_nonblocking_download_ack connection event” on page 375.
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].

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.

● 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.

See also
● “ml_add_missing_dnld_scripts system procedure” on page 617

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.

```sql
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 on the consolidated database. The use of question
marks has been deprecated in SQL scripts. The following script shows how you can use named
parameters:

```sql
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 301
- “Partitioned rows among remote databases” on page 104
- “download_delete_cursor scripts” on page 244

**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 617.

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 database via a download_delete_cursor, the child records are automatically deleted as well. This is not applicable to Blackberry devices.

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.

```sql
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:

```sql
SELECT CAST( NULL AS INTEGER ), CAST( NULL AS INTEGER ) FROM SYSIBM.SYSDUMMY1
```

For Oracle consolidated databases, you can use the following syntax:

```sql
SELECT NULL, NULL FROM DUAL
```
See also

For more information about using download_delete_cursor scripts, see:

● “download_cursor table event” on page 301
● “download_delete_cursor table event” on page 303
● “Deletes” on page 122
● “Temporarily stopping the synchronization of deletes” on page 123
● “STOP SYNCHRONIZATION DELETE statement [MobiLink]” [SQL Anywhere Server - SQL Reference]
● “Partitioned rows among remote databases” on page 104
● “Snapshot synchronization” on page 102

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.

```sql
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.

```sql
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].

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 also**

- “handle_error connection event” on page 344
- “handle_odbc_error connection event” on page 349
- “report_error connection event” on page 378
- “report_odbc_error connection event” on page 381


**Error reporting**

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 344
- “handle_odbc_error connection event” on page 349
- “report_error connection event” on page 378
- “report_odbc_error connection event” on page 381

**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.

```sql
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 “Scripts to upload rows” on page 239.

**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 “Scripts to download rows” on page 242.

**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.
This transaction may not be performed on the same database connection as the other events for the target synchronization so 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 pseudocode 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 255.

For the details of download processing, see “Events during download” on page 257.
Events during upload

The following pseudocode illustrates how upload events and upload scripts are invoked. The pseudocode uses the following conventions:

- **A script is defined as a real script** 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 define it as an ignored 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 248.

Overview of the 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.

if( upload_insert script is real ) {
  upload_insert
} 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 or upload_fetch_column_conflict 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 and upload_fetch_column_conflict scripts are not
      defined ) {
      if( ConflictsAreExpected ) {
        error
        } 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_fetch script cannot defined as an ignored script
        error
      }
    } 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.
if( upload_delete is real ) {
    upload_delete
} 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 248.

------------------------------------------------------
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 specifies 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 data scripts can reference the named parameters for column values.

The following events have data scripts associated with them:
● “download_cursor table event”
● “download_delete_cursor table event”
● “handle_UploadData connection event”
● “handle_DownloadData connection 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_update table event”

### Java and .NET data scripts returning SQL (removed)

Starting in version 16, the ability for Java and .NET scripting logic to return strings that are interpreted by MobiLink server as SQL scripts has been removed in all 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 can be updated. The first example uses SQL and the second example does not.

```java
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();
    PreparedStatement 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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.file_authentication_code</td>
<td>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.</td>
<td>1</td>
</tr>
<tr>
<td>s.filename</td>
<td>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 mlsrv16 -ftr or -fru 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.</td>
<td>2</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>3</td>
</tr>
<tr>
<td>s.subdir</td>
<td>VARCHAR(128). 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.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.remote_key</td>
<td>VARCHAR(128). Optional IN parameter to specify a remote key for the file transfer.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>
### 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

- “Script additions and deletions” on page 235
- “-ftr mlsrv16 option” on page 58
- “-ftru mlsrv16 option” on page 58
- “MobiLink File Transfer utility (mlfiletransfer)” [MobiLink - Client Administration]
- “MLFileDownload method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- UltraLite: “MobiLink file transfers” [UltraLite - Database Management and Reference]
- “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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the Mobi-Link server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Parameter name for SQL scripts</td>
<td>Description</td>
<td>Order (deprecated for SQL)</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>s.file_authentication_code</td>
<td>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.</td>
<td>1</td>
</tr>
<tr>
<td>s.filename</td>
<td>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 mlsrv16 -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.</td>
<td>2</td>
</tr>
<tr>
<td>s.file_size</td>
<td>INTEGER. This optional IN parameter can be used to limit the size of file that can be uploaded.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>3</td>
</tr>
<tr>
<td>s.subdir</td>
<td>VARCHAR(128). 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.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.remote_key</td>
<td>VARCHAR(128). Optional IN parameter to specify a remote key for the file transfer.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

**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

- “Script additions and deletions” on page 235
- “-ftr mlsrv16 option” on page 58
- “-ftru mlsrv16 option” on page 58
- “MobiLink File Transfer utility (mlfiletransfer)” [MobiLink - Client Administration]
- “MLFileUpload method [UltraLite Embedded SQL]” [UltraLite - C and C++ Programming]
- UltraLite: “MobiLink file transfers” [UltraLite - Database Management and Reference]
- “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration]

**authenticate_parameters connection event**

Receives values from the remote database 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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.authentication_status</td>
<td>INTEGER. This is an INOUT parameter.</td>
<td>1</td>
</tr>
<tr>
<td>s.authentication_message</td>
<td>VARCHAR(1024). This is an INOUT parameter. Provides an authentication message.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>2</td>
</tr>
</tbody>
</table>
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:

<table>
<thead>
<tr>
<th>Returned Value</th>
<th>authentication_status</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V &lt;= 1999</td>
<td>1000</td>
<td>Authentication succeeded.</td>
</tr>
<tr>
<td>1000 &lt;= V &lt;= 2999</td>
<td>2000</td>
<td>Authentication succeeded, but password expiring soon.</td>
</tr>
<tr>
<td>2000 &lt;= V &lt;= 3999</td>
<td>3000</td>
<td>Authentication failed as password has expired.</td>
</tr>
<tr>
<td>4000 &lt;= V &lt;= 4999</td>
<td>4000</td>
<td>Authentication failed.</td>
</tr>
<tr>
<td>5000 &lt;= V &lt;= 5999</td>
<td>5000</td>
<td>Unable to authenticate because the remote ID is already in use. Try the synchronization again later.</td>
</tr>
<tr>
<td>6000 &lt;= V</td>
<td>4000</td>
<td>If the returned value is greater than 5999, MobiLink interprets it as a returned value of 4000 (authentication failed).</td>
</tr>
</tbody>
</table>

- **authentication_message**  This optional parameter provides an authentication message.

This named parameter is initialized to NULL before its first use by a user authentication script. Its returning message is then passed into the next user authentication script, if the script takes this named parameter. The final message is translated into the character set of the remote database.

If no error occurred during execution of the user authentication scripts, this message is then sent to the client by the MobiLink server before precessing the upload stream, regardless of the user authentication status.

This message is sent to the client, even if the user authentication failed.

- **remote_ID**  The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.

See “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration].
● **script_version**  This optional parameter specifies that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.

● **username**  This parameter is the MobiLink user name. VARCHAR(128).

● **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 216
- “Script additions and deletions” on page 235
- “Authentication parameters” on page 231
- “MobiLink users” [MobiLink - Client Administration]
- “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration]
- “Custom user authentication” [MobiLink - Client Administration]
- “authenticate_user connection event” on page 266
- “authenticate_user_hashed connection event” on page 271
- “begin_synchronization connection event” on page 287
- 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:

```c
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:

```sql
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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and
“SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset
of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.authentication_status</td>
<td>INTEGER. This is an INOUT parameter.</td>
<td>1</td>
</tr>
</tbody>
</table>
### 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:

---

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.authentication_message</td>
<td>VARCHAR(1024). This is an INOUT parameter. Provides an authentication message.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>2</td>
</tr>
<tr>
<td>s.password</td>
<td>VARCHAR(128). The password for authentication purposes. If the user does not supply a password, this value is null.</td>
<td>3</td>
</tr>
<tr>
<td>s.new_password</td>
<td>VARCHAR(128). The new password, if this is being used to reset the password. If the user does not change their password, this value is null.</td>
<td>4</td>
</tr>
<tr>
<td>s.new_remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID, if the remote ID is new in the consolidated database. If the remote ID is not new, the value is null.</td>
<td></td>
</tr>
<tr>
<td>s.new_username</td>
<td>VARCHAR(128). The MobiLink user name, if the username is new in the consolidated database. If the user name is not new, the value is null.</td>
<td></td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>
### Returned Value | authentication_status | Description
--- | --- | ---
V <= 1999 | 1000 | Authentication succeeded.
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.
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*]

#### authentication_message
This optional parameter provides an authentication message.

This named parameter is initialized to NULL before its first use by a user authentication script. Its returning message is then passed into the next user authentication script, if the script takes this named parameter. The final message is translated into the character set of the remote database.

If no error occurred during execution of the user authentication scripts, this message is then sent to the client by the MobiLink server before processing the upload stream, regardless of the user authentication status.

This message is sent to the client, even if the user authentication failed.

#### username
This optional parameter is the MobiLink user name.

See “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.
**new_remote_id**  
This optional parameter indicates a new remote ID. If the remote ID is not new, this is null.

**new_username**  
This optional parameter indicates a new user name. If the user name is not new, this is null.

**script_version**  
This optional parameter specifies that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.

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 “Authentication to external servers” [MobiLink - Client Administration].

**See also**
- “Script parameters” on page 216
- “Script additions and deletions” on page 235
- “MobiLink users” [MobiLink - Client Administration]
- “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration]
- “Custom user authentication” [MobiLink - Client Administration]
- “Authentication to external servers” [MobiLink - Client Administration]
- “authenticate_user_hashed connection event” on page 271
- “authenticate_parameters connection event” on page 263
- “modify_user connection event” on page 366
- “begin_synchronization connection event” on page 287

**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.

```sql
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.

```sql
CREATE PROCEDURE my_auth( inout @auth_status int, in @user_name varchar(128) )
```

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0 269
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.

```java
public void authenticateUser(
   ianywhere.ml.script.InOutInteger authStatus,
   String user,
   String pwd,
   String newPwd )
   throws java.sql.SQLException {
   _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 {
         // Authentication failed.
         authStatus.setValue( 1000 );
      }
   } else {
      // Authentication failed.
      authStatus.setValue( 4000 );
   }
}
```
.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.

```sql
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.

```csharp
namespace TestScripts {
    public class Test {
        string _curUser = null;
        public void AuthUser(
            ref int authStatus,
            string user,
            string pwd,
            string newPassword ) {
            // A real authenticate_user handler would
            // handle more authentication code states.
            _curUser = user;
            if( CheckPwd( user, pwd ) ) {
                // Authentication successful.
                if( newPassword != null ) {
                    // Password is being changed.
                    if( ChangePwd( user, pwd, newPassword ) ) {
                        // Authentication OK and password change OK.
                        authStatus = 1001;
                    } else {
                        // Authentication OK but password change failed.
                        System.Console.WriteLine( "user: " + user + " pwd change failed!" );
                        authStatus = 1002;
                    } else {
                        // Authentication failed.
                        authStatus = 1000;
                    }
                } else {
                    authStatus = 4000;
                }
            } else {
                authStatus = 1000;
            }
        }
    }
}
```

For a more detailed example of an authenticate_user script written in C# in .NET, see “.NET synchronization example” on page 510.

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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.authentication_status</td>
<td>INTEGER. This is an INOUT parameter.</td>
<td>1</td>
</tr>
<tr>
<td>s.authentication_message</td>
<td>VARCHAR(1024). This is an INOUT parameter. Provides an authentication message.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>2</td>
</tr>
<tr>
<td>shashed_password</td>
<td>BINARY(32). If the user does not supply a password, this value is null.</td>
<td>3</td>
</tr>
<tr>
<td>shashed_new_password</td>
<td>BINARY(32). If this event is not being used to change the user's password, this value is null.</td>
<td>4</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

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_userhashed_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 216
- “Script additions and deletions” on page 235
- “MobiLink users” [MobiLink - Client Administration]
- “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration]
- “Custom user authentication” [MobiLink - Client Administration]
- “authenticate_user connection event” on page 266
- “authenticate_parameters connection event” on page 263

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.

```sql
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.

```sql
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.

```sql
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.

```java
public void 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 {
            // Authorization failed.
            authStatus.setValue( 1000 );
        }
    } else {
        // Authorization failed.
        authStatus.setValue( 4000 );
    }
}
```

.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.

```sql
CALL ml_add_dnet_connection_script('ver1', 'authenticate_user_hashed', 'TestScripts.Test.AuthUserHashed')
```

The following is the sample .NET method AuthUserHashed.

```csharp
namespace TestScripts {
    public class Test {
```
```java
    string _curUser = null;
    public void 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( CheckPwHashed( user, pwdHash ) )
    // Authorization successful.
    if( newPwHash != null )
    // Password is being changed.
    if( ChangePwdHashed( user, pwdHash, newPwHash ) )
    // 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;
    }}}
```

### 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.
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

● “Script additions and deletions” on page 235
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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.last_download</td>
<td>TIMESTAMP. The oldest download time of any synchronized table.</td>
<td>1</td>
</tr>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>2</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

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 216
- “Script additions and deletions” on page 235
- “end_download connection event” on page 313
- “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration]
- “Last download times in scripts” on page 100
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 void beginDownloadConnection(
   Timestamp ts,
   String user
) throws java.sql.SQLException {
   prepDeleteTables ( _syncConn, ts, user );
}
```

.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 void BeginDownload(
   DateTime timestamp,
   string user ) {
   prepDeleteTables ( _syncConn, ts, user );
}
```

**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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.last_table_download</td>
<td>TIMESTAMP. The last download time for the table.</td>
<td>1</td>
</tr>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>2</td>
</tr>
<tr>
<td>s.table</td>
<td>VARCHAR(128). The table name.</td>
<td>3</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

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 216
- “Script additions and deletions” on page 235
- “end_download table event” on page 315
- “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration]
- “Last download times in scripts” on page 100
**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 16 consolidated database.

```sql
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.

```sql
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`.

```java
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. (Printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```java
package ExamplePackage;
public class ExampleClass { 
 String _curUser = null;
 public void beginDownloadTable( 
 String user, 
 String table ) { 
 java.lang.System.out.println("Beginning to process download for: " + 
 table);
 } 
}
```

**.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`.

```csharp
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. (Printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```csharp
namespace TestScripts {
    public class Test {
        string _curUser = null;
        public void BeginTableDownload(
            string user,
            string table ) {
            System.Console.WriteLine("Beginning to process download for: " + table);
        }
    }
}
```

### 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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.last_table_download</td>
<td>TIMESTAMP. The last download time for the table.</td>
<td>1</td>
</tr>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>2</td>
</tr>
<tr>
<td>s.table</td>
<td>VARCHAR(128). The table name.</td>
<td>3</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>
MobiLink events

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 216
- “Script additions and deletions” on page 235
- “begin_download_rows table event” on page 282
- “end_download_rows table event” on page 319
- “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration]
- “Last download times in scripts” on page 100

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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.
### Synchronization events

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.last_table_download</td>
<td>TIMESTAMP. The last download time for the table.</td>
<td>1</td>
</tr>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>2</td>
</tr>
<tr>
<td>s.table</td>
<td>VARCHAR(128). The table name.</td>
<td>3</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

**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 216
- “Script additions and deletions” on page 235
- “begin_download_deletes table event” on page 281
- “end_download_deletes table event” on page 318
- “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration]
- “Last download times in scripts” on page 100

### 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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.generation_number</td>
<td>INTEGER. This is an INOUT parameter. If your deployment does not use file-based downloads, this parameter can be ignored. The default is 1.</td>
<td>1</td>
</tr>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>2</td>
</tr>
<tr>
<td>s.publication_name</td>
<td>VARCHAR(128). The name of the publication.</td>
<td>3</td>
</tr>
<tr>
<td>s.last_publication_upload</td>
<td>TIMESTAMP. The time of the last successful upload of this publication.</td>
<td>4</td>
</tr>
<tr>
<td>s.last_publication_download</td>
<td>TIMESTAMP. The last download time for the publication.</td>
<td>5</td>
</tr>
<tr>
<td>s.subscription_id</td>
<td>VARCHAR(128). The remote subscription ID.</td>
<td>6</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

Default action

The default generation number is 1. If no script is defined for this event, the generation number sent to the remote database 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 database 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 216
- “Script additions and deletions” on page 235
- “end_publication connection event” on page 321
- “MobiLink file-based download” on page 195
- “MobiLink generation numbers” on page 201
- “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration]
- “Last download times in scripts” on page 100

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.

```sql
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.

    package ExamplePackage;
    public class ExampleClass{
        java.util.ArrayList<String> _publicationNames;
        int                _numPublications = 0;

        public void beginPublication( ianywhere.ml.script.InOutInteger generation_number, String user, String pub_name ){
            _numPublications++;
            _publicationNames.add( pub_name );
        }
    }

.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.

    using System.Collections.Generic;
    namespace TestScripts
    {
        class Test
        {
            List<string>    _publicationNames = new List<string>();
            int        _numPublications  = 0;

            public void BeginPub( ref int    generation_number, string user, string pub_name ){
                _numPublications++;
                _publicationNames.Add( pub_name );
            }
        }
    }
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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>1</td>
</tr>
<tr>
<td>s.new_remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID, if the remote ID is new in the consolidated database. If the remote ID is not new, the value is null.</td>
<td></td>
</tr>
<tr>
<td>s.new_username</td>
<td>VARCHAR(128). The MobiLink user name, if the user name is new in the consolidated database. If the user name is not new, the value is null.</td>
<td></td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

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 216
- “Script additions and deletions” on page 235
- “end_synchronization connection event” on page 324
- “begin_synchronization table event” on page 289
- “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 are going to reference that value many times in subsequent scripts.

```sql
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.

```java
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.

```java
package ExamplePackage;
public class ExampleClass {
    String _curUser = null;
    public void beginSynchronizationConnection(String user) {
        _curUser = user;
    }
}
```

.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.

```csharp
CALL ml_add_dnet_connection_script( 'ver1',
    'begin_synchronization',
```
The following is the sample .NET method BeginSync. It saves the name of the synchronizing user for later use.

```csharp
namespace TestScripts {
    public class Test {
        string _curUser = null;
        public void BeginSync(string user) {
            _curUser = user;
        }
    }
}
```

### 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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>1</td>
</tr>
<tr>
<td>s.table</td>
<td>VARCHAR(128). The table name.</td>
<td>2</td>
</tr>
<tr>
<td>s.new_remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID, if the remote ID is new in the consolidated database. If the remote ID is not new, the value is null.</td>
<td>2</td>
</tr>
<tr>
<td>Parameter name for SQL scripts</td>
<td>Description</td>
<td>Order (deprecated for SQL)</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>s.new_username</td>
<td>VARCHAR(128). The MobiLink user name, if the user name is new in the consolidated database. If the user name is not new, the value is null.</td>
<td></td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

**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 216
- “Script additions and deletions” on page 235
- “end_synchronization table event” on page 326
- “begin_synchronization connection event” on page 287
- “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.

```sql
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.

```java
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.

```java
package ExamplePackage;
import java.util.ArrayList;
import java.sql.Timestamp;
class ExampleClass
{
    ArrayList<String> _tableList;
    String _curTable;
    public void beginSynchronizationTable(String user, String table )
    {
        _curTable = table;
        _tableList.add( table );
    }

    public void endTableDownload(Timestamp ts, String user, String table )
    {
        _curTable = 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.

```csharp
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.

```csharp
using System.Collections.Generic;
using System;
namespace TestScripts
{
    class Test
    {
    List<string> _tableList = new List<string>();
    string _curTable = "";
```
```java
public void BeginSynchronizationTable( string user, 
        string table )
{
    _curTable = table;
    _tableList.Add( table );
}

public void EndTableDownload( DateTime timestamp, 
        string user, 
        string table )
{
    _curTable = 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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The Mobi-Link remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The Mobi-Link user name.</td>
<td>1</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>
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 216
- “Script additions and deletions” on page 235
- “end_upload connection event” on page 329
- “begin_upload table event” on page 294
- “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.

```sql
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.

```sql
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. (Printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```java
package ExamplePackage;
public class ExampleClass {
    String _curUser = null;
    public void beginUploadConnection( String user ) {
        java.lang.System.out.println(
            "Starting upload for user: " + user);
    }
}
```
.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.

```sql
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.

```csharp
namespace TestScripts {
    public class Test {
        string _curUser = null;
        public void BeginUpload( string curUser ) {
            _curUser = curUser;
        }
    }
}
```

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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>1</td>
</tr>
<tr>
<td>s.table</td>
<td>VARCHAR(128). The table name.</td>
<td>2</td>
</tr>
</tbody>
</table>
### 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 216
- “Script additions and deletions” on page 235
- “end_upload table event” on page 331
- “begin_upload connection event” on page 292
- “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. You can delete from a global temporary table in this event, in preparation for receiving the new rows.

```sql
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.

```java
CALL ml_add_java_table_script(  
'verl',  
'table1',  
'begin_upload',  
'ExamplePackage.ExampleClass.beginUploadTable' )
```
The following is the sample Java method `beginUploadTable`. It prints a message to the MobiLink message log. (Printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```java
package ExamplePackage;
public class ExampleClass {
    String _curUser = null;
    public void beginUploadTable(
        String user,
        String table ) {
        java.lang.System.out.println("Beginning to process upload for: "+ table);
    }
}
```

**.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`.

```sql
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. (Printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```csharp
namespace TestScripts {
    public class Test {
        string _curUser = null;
        public void BeginTableUpload(
            string user,
            string table ) {
            System.Console.WriteLine("Beginning to process upload for: "+ table);
        }
    }
}
```

### 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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.
### 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 216
- “Script additions and deletions” on page 235
- “end_upload_deletes table event” on page 334
- “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.

```sql
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,
');
```
MobiLink events

```sql
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`.

```java
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. (Printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```java
package ExamplePackage;
public class ExampleClass {
  String _curUser = null;
  public void beginUploadDeletes(
    String user,
    String table )
    throws java.sql.SQLException {
    java.lang.System.out.println(
      "Starting upload deletes for table: " + table );
  }
}
```

.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`.

```csharp
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. (Printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```csharp
namespace TestScripts {
  public class Test {
    string _curUser = null;
    public void BeginUploadDeletes(
      string user,
      string table ) {
      System.Console.WriteLine(
        "Starting upload deletes for table: " + table );
    }
  }
}
```
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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>1</td>
</tr>
<tr>
<td>s.table</td>
<td>VARCHAR(128). The table name.</td>
<td>2</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

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 216
● “Script additions and deletions” on page 235
● “end_upload_rows table event” on page 336
● “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. (Printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```
package ExamplePackage;
public class ExampleClass {
    String _curUser = null;
    public void beginUploadRows(
        String user,
        String table )
    throws java.sql.SQLException {
        java.lang.System.out.println("Starting upload rows for table: " +
            table + " and user: " + user );
    }
}
```

.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. (Printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```csharp	namespace TestScripts {
    public class Test {
        string _curUser = null;
        public void BeginUploadRows(string user, string table) {
            System.Console.WriteLine(
                "Starting upload rows for table: " + table + " and user: " + user);
        }
    }
}
```

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 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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.last_table_download</td>
<td>TIMESTAMP. The last download time for the table.</td>
<td>1</td>
</tr>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>2</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>
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:

\[
\text{AND last_modified} \geq \{\text{ml s.last_table_download}\}
\]

This script must be implemented in SQL. For Java or .NET processing of rows, see “Direct row handling” on page 585.

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 258
- “Script parameters” on page 216
- “Script additions and deletions” on page 235
- “Scripts to download rows” on page 242
- “download_cursor scripts” on page 243
- “Partitioned rows among remote databases” on page 104
- “download_delete_cursor table event” on page 303
- “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration]
- “Last download times in scripts” on page 100
- "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.

```sql
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 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 are going 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

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.last_table_download</td>
<td>TIMESTAMP. The last download time for the table.</td>
<td>1</td>
</tr>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink username.</td>
<td>2</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

### 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 the MobiLink server tells the remote database to delete all the rows in the table. See “Deleting all the rows in a table” on page 245.

Rows deleted from the consolidated database cannot 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:

```sql
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 585.
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 258
- “Script parameters” on page 216
- “Script additions and deletions” on page 235
- “download_cursor table event” on page 301
- “Scripts to download rows” on page 242
- “Partitioned rows among remote databases” on page 104
- “download_delete_cursor scripts” on page 244
- “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration]
- “Last download times in scripts” on page 100
- "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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 script. 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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name as specified in your SYNCHRONIZATION USER definition.</td>
<td>1</td>
</tr>
<tr>
<td>s.warnings</td>
<td>INTEGER. The number of warnings issued.</td>
<td>2</td>
</tr>
<tr>
<td>s.errors</td>
<td>INTEGER. The number of errors, including handled errors, that occurred.</td>
<td>3</td>
</tr>
<tr>
<td>s.fetched_rows</td>
<td>INTEGER. The number of rows fetched by the download_cursor script.</td>
<td>4</td>
</tr>
<tr>
<td>s.deleted_rows</td>
<td>INTEGER. The number of rows fetched by the download_delete_cursor script.</td>
<td>5</td>
</tr>
<tr>
<td>s.filtered_rows</td>
<td>INTEGER. The number of rows from the fetched_rows parameter actually sent to the remote. This reflects download filtering of uploaded values.</td>
<td>6</td>
</tr>
<tr>
<td>s.bytes</td>
<td>INTEGER. The number of bytes sent to the remote database as the download.</td>
<td>7</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

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 216
- “Script additions and deletions” on page 235
- “download_statistics table event” on page 308
- “upload_statistics connection event” on page 413
- “upload_statistics table event” on page 417
- “synchronization_statistics connection event” on page 388
- “synchronization_statistics table event” on page 392
- “time_statistics connection event” on page 395
- “time_statistics table event” on page 398
- “MobiLink Profiler” on page 176
- “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.

```sql
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.

```java
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. (Printing the number of fetched rows to the MobiLink message log might be useful at development time but would slow down a production server.)

```java
package ExamplePackage;
public class ExampleClass {
    String _curUser = null;
pUBLIC void 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 );
        }
}
```

**.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.

```sql
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. (Printing the number of fetched rows to the MobiLink message log might be useful at development time but would slow down a production server.)

```csharp
namespace TestScripts {
pUBLIC class Test {
    string _curUser = null;
pUBLIC void 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 );
        }
}
```

**download_statistics table event**

Provides access to synchronization statistics for download operations by table.
In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name as specified in your SYNCHRONIZATION USER definition.</td>
<td>1</td>
</tr>
<tr>
<td>s.table</td>
<td>VARCHAR(128). The table name.</td>
<td>2</td>
</tr>
<tr>
<td>s.warnings</td>
<td>INTEGER. The number of warnings issued.</td>
<td>3</td>
</tr>
<tr>
<td>s.errors</td>
<td>INTEGER. The number of errors, including handled errors, that occurred.</td>
<td>4</td>
</tr>
<tr>
<td>s.fetched_rows</td>
<td>INTEGER. The number of rows fetched by the download_cursor script.</td>
<td>5</td>
</tr>
<tr>
<td>s.deleted_rows</td>
<td>INTEGER. The number of rows fetched by the download_delete_cursor script.</td>
<td>6</td>
</tr>
<tr>
<td>s.filtered_rows</td>
<td>INTEGER. The number of rows filtered from the fetched_rows. This reflects download filtering of uploaded values.</td>
<td>7</td>
</tr>
<tr>
<td>s.bytes</td>
<td>INTEGER. The number of bytes sent to the remote database as the download.</td>
<td>8</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>
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 216
- “Script additions and deletions” on page 235
- “download_statistics connection event” on page 305
- “upload_statistics connection event” on page 413
- “upload_statistics table event” on page 417
- “synchronization_statistics connection event” on page 388
- “synchronization_statistics table event” on page 392
- “time_statistics connection event” on page 395
- “time_statistics table event” on page 398
- “MobiLink Profiler” on page 176
- “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.

```sql
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. (Printing statistics for a table to the MobiLink message log might be useful at development time but would slow down a production server.)

```java
package ExamplePackage;
public class ExampleClass {
    String _curUser = null;
    public void 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 );
    }
}
```

.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. (Printing statistics for a table to the MobiLink message log might be useful at development time but would slow down a production server.)

```csharp
namespace TestScripts {
    public class Test {
        string _curUser = null;
        public void DownloadTableStats(
            string user,
            string table,
            int warnings,
            int errors,
            int fetchedRows,
            int deletedRows,
            int filteredRows,
```
```csharp
int bytes ) {
    System.Console.WriteLine( "download table stats "
    + "table: " + table + "bytes: " + bytes );
}}}
```

**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 275
- “Script additions and deletions” on page 235

**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.

```sql
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.

```java
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. (Printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```java
package ExamplePackage;
public class ExampleClass {
    String _curUser = null;
p    public void endConnection() {
        java.lang.System.out.println( "Ending connection." );
    }
}
```

**.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.

```sql
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. (Printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```csharp
namespace TestScripts {
p    public class Test {
        string _curUser = null;
p        public void EndConnection() {
            System.Console.WriteLine( "Ending connection." );
        }
    }
}
```

**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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going 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

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.last_download</td>
<td>TIMESTAMP. The oldest download time of any synchronized table.</td>
<td>1</td>
</tr>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td></td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>2</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td></td>
</tr>
</tbody>
</table>

### 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 216
- “Script additions and deletions” on page 235
- “begin_download connection event” on page 277
- “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration]
- “Last download times in scripts” on page 100

### 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.

```sql
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.

```java
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. (Printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```java
package ExamplePackage;
import java.sql.*;
public class ExampleClass {
    String _curUser = null;
    public void endDownloadConnection(Timestamp ts, String user) {
        java.lang.System.out.println( "Ending download for user: " + user );
    }
}
```

.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.

```sql
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. (Printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```csharp
public void EndDownload(DateTime timestamp, string user) {
    System.Console.WriteLine( "Ending download for user: " + user );
}
```

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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.
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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.last_table_download</td>
<td>TIMESTAMP. The last download time for the table.</td>
<td>1</td>
</tr>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>2</td>
</tr>
<tr>
<td>s.table</td>
<td>VARCHAR(128). The table name.</td>
<td>3</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

**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 table-specific, non-statistical action in the download transaction.

You can have one end_download script for each table in the remote database.
See also

- “Script parameters” on page 216
- “Script additions and deletions” on page 235
- “begin_download table event” on page 278
- “end_download connection event” on page 313
- “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration]
- “Last download times in scripts” on page 100

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.

```sql
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.

```java
CALL ml_add_java_table_script ( 
'verl',
'table1',
'end_download',
'ExamplePackage.ExampleClass.endDownloadTable' )
```

The following is the sample Java method endDownloadTable. It resets the current table member variable.

```java
public void endDownloadTable(
Timestamp ts,
String user,
String table ) {
_curTable = 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.

```csharp
CALL ml_add_dnet_table_script(
'verl',
'table1',
'end_download',
'TestScripts.Test.EndTableDownload' )
```

The following is the sample .NET method EndTableDownload. It resets the current table member variable.

```csharp
public void EndTableDownload
(DateTime timestamp,
...
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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.last_table_download</td>
<td>TIMESTAMP. The last download time for the table.</td>
<td>1</td>
</tr>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>2</td>
</tr>
<tr>
<td>s.table</td>
<td>VARCHAR(128). The table name.</td>
<td>3</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
</tr>
</tbody>
</table>

You can have one end_download_deletes script for each table in the remote database.

See also

- “Script parameters” on page 216
- “Script additions and deletions” on page 235
- “begin_download_deletes table event” on page 281
- “end_download_deletes table event” on page 313
- “begin_download_rows table event” on page 282
- “end_download_rows table event” on page 319
- “download_delete_cursor table event” on page 303
- “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration]
- “Last download times in scripts” on page 100

### 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.

<table>
<thead>
<tr>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.</td>
</tr>
</tbody>
</table>

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 are going 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

<table>
<thead>
<tr>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.last_table_download TIMESTAMP. The last download time for the table.</td>
<td>1</td>
</tr>
<tr>
<td>s.remote_id VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username VARCHAR(128). The MobiLink user name.</td>
<td>2</td>
</tr>
<tr>
<td>s.table VARCHAR(128). The table name.</td>
<td>3</td>
</tr>
<tr>
<td>s.script_version VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

### 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.
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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.generation_number</td>
<td>INTEGER. If your deployment does not use file-based downloads, this parameter can be ignored. The default value is 1.</td>
<td>1</td>
</tr>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>2</td>
</tr>
<tr>
<td>s.publication_name</td>
<td>VARCHAR(128). The name of the publication.</td>
<td>3</td>
</tr>
<tr>
<td>s.last_publication_upload</td>
<td>TIMESTAMP. Last successful upload time of this publication.</td>
<td>4</td>
</tr>
<tr>
<td>s.last_publication_download</td>
<td>TIMESTAMP. The last download time of this publication.</td>
<td>5</td>
</tr>
<tr>
<td>Parameter name for SQL scripts</td>
<td>Description</td>
<td>Order (deprecated for SQL)</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>s.subscription_id</td>
<td>VARCHAR(128). The remote subscription ID.</td>
<td>6</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

**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 database 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 216
- “Script additions and deletions” on page 235
- “begin_publication connection event” on page 283
- “MobiLink file-based download” on page 195
- “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration]
- “Last download times in scripts” on page 100
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`.

```sql
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`.

```java
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. (Printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```java
package ExamplePackage;
import java.sql.*;
public class ExampleClass {
    String _curUser = null;
    public void 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 );
    }
}
```

.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`.

```csharp
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. (Printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```csharp
public void EndPub(
    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 );
}
```
```csharp
    string user,
    string pub_name,
    DateTime last_publication_upload,
    DateTime last_publication_download ) {
    System.Console.Write("Finished synchronizing publication " + pub_name );
}
```

## 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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink username.</td>
<td>1</td>
</tr>
<tr>
<td>s.synchronization_ok</td>
<td>INTEGER. This value is 1 for a successful synchronization and 0 for an unsuccessful synchronization.</td>
<td>2</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>
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 database 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 216
● “Script additions and deletions” on page 235
● “begin_synchronization connection event” on page 287
● “begin_synchronization table event” on page 289
● “end_synchronization table event” on page 326
● “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 void endSynchronizationConnection(
  String user )
  throws java.sql.SQLException {
    execUpdate( _syncConn,
        "UPDATE sync_count set count = count + 1 where user_id = " + user )
  }
```
.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.

```csharp
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.

```csharp
namespace TestScripts {
    public class Test {
        string _curUser = null;
        public void EndSync(string user) {
            return(
                "UPDATE sync_count set count = count + 1 where user_id = '" + user + "'");
        }
    }
}
```

**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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>
### Parameter name for SQL scripts

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>1</td>
</tr>
<tr>
<td>s.table</td>
<td>VARCHAR(128). The table name.</td>
<td>2</td>
</tr>
<tr>
<td>s.synchronization_ok</td>
<td>INTEGER. This value is 1 for a successful synchronization and 0 for an unsuccessful synchronization.</td>
<td>3</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

### 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 216
- “Script additions and deletions” on page 235
- “begin_synchronization table event” on page 289
- “end_synchronization connection event” on page 324
- “end_synchronization table event” on page 326
- “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.

```sql
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.

```java
package ExamplePackage;
import ianywhere.ml.script.*;
import java.sql.*;
public class ExampleClass {
    private DBConnectionContext _cc = null;
    public ExampleClass(DBConnectionContext cc) {
        _cc = cc;
    }
    public void endSynchronizationTable() throws SQLException {
        try (Connection conn = _cc.getConnection()) {
            try (PreparedStatement stmt = conn.prepareStatement("DROP TABLE #sales_order")) {
                stmt.executeUpdate();
            }
        }
    }
}
```

.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 EndSynchronizationTable.

```csharp
using iAnywhere.MobiLink.Script;
namespace TestScripts {
    public class ExampleClass {
        private DBConnectionContext _cc = null;
        public ExampleClass(DBConnectionContext cc) {
            _cc = cc;
        }
        public void EndSynchronizationTable() {
            DBConnection conn = _cc.GetConnection();
            try {
                DBCommand cmd = conn.CreateCommand();
                try {
                    cmd.CommandText = "DROP TABLE #sales_order";
                }
            }
        }
    }
}
```
```csharp
    cmd.Prepare();
    cmd.ExecuteNonQuery();
} finally {
    cmd.Close();
}
} finally {
    conn.Close();
}
```

**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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink username.</td>
<td>1</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>
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 216
- “Script additions and deletions” on page 235
- “begin_upload connection event” on page 292
- “end_upload table event” on page 331
- “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration]

SQL example

The following SQL Anywhere SQL script calls the EndUpload stored procedure.

```sql
CALL ml_add_connection_script( 'ver1', '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.

```java
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.

```java
public void 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();
}
```

.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.

```csharp
CALL ml_add_dnet_connection_script( 'ver1', 'end_upload',
```
The following is the sample .NET method `EndUpload`.

```csharp
using iAnywhere.MobiLink.Script;
namespace TestScripts {
    public class ExampleClass {
        DBConnectionContext _cc = null;
        ExampleClass( DBConnectionContext cc ) {
            _cc = cc;
        }
        public void EndUpload( string userName ) {
            DBConnection conn = _cc.GetConnection();
            try {
                DBCommand cmd = conn.CreateCommand();
                try {
                    cmd.CommandText = "CALL EndUpload( ? )";
                    cmd.Prepare();
                    DBParameter parm = new DBParameter();
                    parm.DbType = SQLType.SQL_CHAR;
                    parm.Value = userName;
                    cmd.Parameters.Add( parm );
                    cmd.ExecuteNonQuery();
                } finally {
                    cmd.Close();
                } finally {
                    conn.Close();
                }
            }
        }
    }
}
```

**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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.
### Parameter Description

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink username.</td>
<td>1</td>
</tr>
<tr>
<td>s.table</td>
<td>VARCHAR(128). The table name.</td>
<td>2</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

**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 216
- “Script additions and deletions” on page 235
- “begin_upload table event” on page 294
- “end_upload connection event” on page 329
- “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.

```sql
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.

```sql
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.

```sql
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.

```java
package ExamplePackage;
public class ExampleClass {

  String _curUser = null;

  public void 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.

```sql
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.

```csharp
using iAnywhere.MobiLink.Script;
namespace TestScripts
{

```
public class Test
{
    DBConnection _curConn = null;

    public Test( DBConnectionContext cc )
    {
        _curConn = cc.GetConnection();
    }

    public void 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();
    }
}

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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink username.</td>
<td>1</td>
</tr>
<tr>
<td>s.table</td>
<td>VARCHAR(128). The table name.</td>
<td>2</td>
</tr>
</tbody>
</table>
### 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 216
- “Script additions and deletions” on page 235
- “begin_upload_deletes table event” on page 296
- “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.

```sql
CALL ml_add_table_script(
  'version1',
  'Leads',
  'end_upload_deletes',
  'call EndUploadDeletesLeads()');
```

The following SQL statement creates the EndUploadDeletes stored procedure.

```sql
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 endUploadDeletes. It calls a Java method that manipulates the
database.

    public void endUploadDeletes(
        String user,
        String table
    )
    throws java.sql.SQLException {
        processUploadedDeletes( _syncConn, table );
    }

.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

    CALL ml_add_dnet_table_script(
        'ver1',
        'table1',
        'end_upload_deletes',
        'TestScripts.Test.EndUploadDeletes'
    )

The following is the sample .NET method EndUploadDeletes. It calls a .NET method that manipulates the
database.

    namespace TestScripts {
        public class Test {
            string _curUser = null;
            public void EndUploadDeletes(
                string user,
                string table ) {
                processUploadedDeletes( _syncConn, table );
            }
        }
    }

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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and
“SQL-.NET data types” on page 501.
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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink username.</td>
<td>1</td>
</tr>
<tr>
<td>s.table</td>
<td>VARCHAR(128). The table name.</td>
<td>2</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

**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**

- “Script additions and deletions” on page 235
- “begin_upload_rows table event” on page 299
- “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.

```sql
CALL ml_add_table_script( 'version1',
```
The following is the sample SQL method `EndUploadRows`. It calls a SQL method that manipulates the database.

```sql
CREATE PROCEDURE EndUploadRows (
  IN username VARCHAR(128)
  IN tablename VARCHAR(128) )
BEGIN
  CALL decide_what_to_do(tablename);
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`.

```java
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.

```java
public void endUploadRows(
  String user,
  String table )
  throws java.sql.SQLException {
  processUploadedRows( _syncConn, table );
}
```

**.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`.

```csharp
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.

```csharp
public void EndUploadRows(
  string user,
  string table ) { 
  processUploadedRows( _syncConn, table );
})
```
**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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.next_last_download</td>
<td>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.</td>
<td>1</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>2</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

**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 216
- “Script additions and deletions” on page 235
- “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration]
- “Last download times in scripts” on page 100
- “How download timestamps are generated and used” on page 101
- “modify_next_last_download_timestamp connection event” on page 363
- “modify_last_download_timestamp connection event” on page 360

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:

1. Assume you have a sync table called my_table that is defined as follows:

   ```sql
   CREATE TABLE my_table (  pk              INT PRIMARY KEY NOT NULL,
                           cl              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:

   ```sql
   CREATE PROCEDURE GenerateNextDownloadTimestamp ( p_ts IN OUT TIMESTAMP )
   AS BEGIN
   ```
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 585.

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 [MobiLink server Java]” on page 440.

For .NET, see “DBConnectionContext interface [MobiLink server .NET]” on page 524.

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. 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 258
- “Direct row handling” on page 585
- “Direct downloads” on page 594
- “DownloadData interface [MobiLink server Java]” on page 445
- “DownloadTableData interface [MobiLink server Java]” on page 448
- “DownloadData interface [MobiLink server .NET]” on page 545
- “DownloadTableData interface [MobiLink server .NET]” on page 547
- “handle_UploadData connection event” on page 352
- “Required scripts” on page 235
- “Script additions and deletions” on page 235

**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.

```java
CALL ml_add_java_connection_script( 'ver1', 'handle_DownloadData', 'MyPackage.MobiLinkOrders.handleDownload' )
```

See “ml_add_java_connection_script system procedure” on page 612.

The following example shows you how to use the handleDownload method to create a download.

The following code sets up a class level DBCConnectionContext instance in the constructor for a class called MobiLinkOrders.

```java
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");

    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;

amespace 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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.action_code</td>
<td>INTEGER. This is an INOUT parameter. Set this value to tell MobiLink server how to respond to the error.</td>
<td>1</td>
</tr>
<tr>
<td>s.error_code</td>
<td>INTEGER. The native RDBMS error code.</td>
<td>2</td>
</tr>
<tr>
<td>s.error_message</td>
<td>TEXT. The native RDBMS error message.</td>
<td>3</td>
</tr>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink username.</td>
<td>4</td>
</tr>
<tr>
<td>s.table</td>
<td>VARCHAR(128). The table whose script had the error. If the script is not a table script, the table name is null.</td>
<td>5</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

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:
MobiLink events

- **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 mlsrv16 option” on page 71.

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:

  ```sql
  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:

  ```sql
  {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 216
- “Script additions and deletions” on page 235
- “report_error connection event” on page 378
- “report_odbc_error connection event” on page 381
- “handle_odbc_error connection event” on page 349
- “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration]
- “Data scripts” on page 258

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.

```java
package ExamplePackage;
public class ExampleClass {
    public void handleError( ianywhere.ml.script.InOutInteger actionCode,
                            int errorCode,
                            String errorMessage,
                            String user,
                            String table ) {
        // -196 is SQLE_INDEX_NOT_UNIQUE
        // -194 is SQLE_INVALID_FOREIGN_KEY
        if( errorCode == -196 || errorCode == -194 ) {
            // ignore the error and keep going
            actionCode.setValue( 1000 );
        } else {
            // abort the synchronization
            actionCode.setValue( 3000 );
        }
    }
}
```

.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.

```sql
CALL ml_add_dnet_connection_script('ver1', 'handle_error', 'TestScripts.Test.HandleError')
```

The following is the sample .NET method HandleError.

```csharp
namespace TestScripts {
    public class Test {
        public void HandleError( ref int actionCode,
                                 int errorCode,
                                 string errorMessage,
                                 string user,
                                 string table ) {
            // -196 is SQLE_INDEX_NOT_UNIQUE
            // -194 is SQLE_INVALID_FOREIGN_KEY
            if( errorCode == -196 || errorCode == -194 ) {
                // ignore the error and keep going
                actionCode = 1000;
            } else {
                // abort the synchronization
                actionCode = 3000;
            }
        }
    }
}
```
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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.action_code</td>
<td>INTEGER. This is an INOUT parameter. Set this value to tell the MobiLink server how to respond to the error.</td>
<td>1</td>
</tr>
<tr>
<td>s.odbc_state</td>
<td>VARCHAR(5). The ODBC SQLSTATE.</td>
<td>2</td>
</tr>
<tr>
<td>s.error_message</td>
<td>TEXT. The ODBC error message</td>
<td>3</td>
</tr>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>4</td>
</tr>
<tr>
<td>s.table</td>
<td>VARCHAR(128). The table name.</td>
<td>5</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

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.
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 216
- “Script additions and deletions” on page 235
- “handle_error connection event” on page 344
- “report_error connection event” on page 378
- “report_odbc_error connection event” on page 381
- “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 HandleODBCSError stored procedure to the handle_odbc_error event.

```sql
CALL ml_add_connection_script(
  'ver1',
  'handle_odbc_error',
  'CALL HandleODBCSError(
    {ml s.action_code},
    {ml s.ODBC_state},
    {ml s.error_message},
    {ml s.username},
    {ml s.table} )' )
```

The following SQL statement creates the HandleODBCSError stored procedure.

```sql
CREATE PROCEDURE HandleODBCSError(
  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.

package ExamplePackage;
public class ExampleClass
{
  public void handleODBCError( ianywhere.ml.script.InOutInteger 
      actionCode,
      String            odbcState,
      String            errorMessage,
      String            user,
      String            table )
  {
    if( odbcState == "23000" ) {
      // Ignore the error and keep going.
      actionCode.setValue( 1000 );
    } else {
      // Abort the synchronization.
      actionCode.setValue( 3000 );
    }
  }
}

.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.

namespace TestScripts
{
}
public class Test {
    public void HandleODBCError( ref int    actionCode,
                              string    odbcState,
                              string    errorMessage,
                              string    user,
                              string    table )
    {
        if( odbcState == "23000" ) {
            // Ignore the error and keep going.
            actionCode = 1000;
        } else {
            // Abort the synchronization.
            actionCode = 3000;
        }
    }
}

handle_UploadData connection event

A non-SQL data script used by direct row handling to process uploaded rows.

Parameters

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>UploadData</td>
<td>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.</td>
<td>1</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

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 585.

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 “Direct uploads” on page 588.

Column names are always sent on the first synchronization to a MobiLink server instance by default, then cached by MobiLink server to avoid re-sending. Optionally, you can establish column names using the ml_add_column system procedure (deprecated). Otherwise you can refer to columns by index, as defined at the remote database.

To get the uploaded pre-image columns for an update, use the SetOldRowValues and SetNewRowValues methods. See “Direct upload conflicts” on page 589.

Note
This event cannot be implemented as SQL.

See also
- “Data scripts” on page 258
- “Direct row handling” on page 585
- “Direct uploads” on page 588
- “UploadData interface [MobiLink server Java]” on page 489
- “UploadedTableData interface [MobiLink server Java]” on page 491
- “UploadData interface [MobiLink server .NET]” on page 573
- “UploadedTableData interface [MobiLink server .NET]” on page 575
- “handle_DownloadData connection event” on page 341
- “Required scripts” on page 235
- “Script additions and deletions” on page 235

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.

```java
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 612.

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.

```java
package MyPackage;
import ianywhere.ml.script.*;
import java.sql.*;
```

```java
```
import java.io.*;
// ...
public class MyClass {
    String _curUser = null;
    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() ) {
            // 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 were
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. The printRow method prints out each row in a result set.

```java
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.

```java
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.

```csharp
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 ) {
```

MobiLink events

356 Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
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;
    }
    Console.Write( row + " );
}
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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.error_message</td>
<td>VARBINARY(1024). This is an IN-OUT parameter, representing the error message.</td>
<td>1</td>
</tr>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>2</td>
</tr>
<tr>
<td>s.error_code</td>
<td>INTEGER. The MobiLink error code associated with the error_message.</td>
<td>3</td>
</tr>
</tbody>
</table>
### 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 216
- “Script additions and deletions” on page 235
- “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:

```sql
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:

```sql
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.

```java
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.

```java
package ExamplePackage;
public class ExampleClass {
    String _curUser = null;
    public void 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) );
    }
}
```

.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.

```java
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.

```java
namespace TestScripts {
    public class Test {
        string _curUser = null;
        public void ModifyLastErrorMessage(ref string errorMessage,
            string userName,
            string errorCode  ) {
            System.Console.WriteLine( "error message: " + errorMessage );
            System.Console.WriteLine( "error code: " +  errorCode );
        }
    }
}
```

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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.last_download</td>
<td>TIMESTAMP. The oldest download time for any synchronized table. This is an INOUT parameter.</td>
<td>1</td>
</tr>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>2</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

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 216
- “Script additions and deletions” on page 235
- “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration]
- “Last download times in scripts” on page 100
- “How download timestamps are generated and used” on page 101
- “modify_next_last_download_timestamp connection event” on page 363
- “generate_next_last_download_timestamp event” on page 339

SQL example

The following SQL statement creates a stored procedure. The following syntax is for Oracle consolidated databases. When creating a stored procedure in Oracle that takes in a parameter and also passes out the parameter, ensure that the parameter is marked as IN OUT, as shown below:

```sql
CREATE OR REPLACE PROCEDURE ModifyDownloadTimestamp (
  download_timestamp  IN OUT TIMESTAMP,
  user_name   IN VARCHAR
) AS
BEGIN
  -- N is the maximum replication latency in consolidated cluster
  download_timestamp := download_timestamp - 1;
END;
```

The following syntax is for SQL Anywhere, Adaptive Server Enterprise, and SQL Server consolidated databases:

```sql
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:

```sql
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.

```java
public void 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() );
}
```

.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.

```csharp
public void 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() );
}
```

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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.next_last_download</td>
<td>TIMESTAMP. This is an INOUT parameter. The MobiLink server generates this value immediately after the upload is committed.</td>
<td>1</td>
</tr>
<tr>
<td>s.last_download</td>
<td>TIMESTAMP. This is the last download time for the current synchronization.</td>
<td>2</td>
</tr>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>3</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

**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, 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 216
- “Script additions and deletions” on page 235
- “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration]
- “Last download times in scripts” on page 100
- “How download timestamps are generated and used” on page 101
- “modify_last_download_timestamp connection event” on page 360
- “generate_next_last_download_timestamp event” on page 339

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:

```sql
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:

```sql
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.

```sql
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.

```java
public void modifyNextDownloadTimestamp(
   Timestamp NextLastDownload,
   Timestamp lastDownload,
   String userName ) {
   NextLastDownload.setHours( NextLastDownload.getHours() -1 );
}
```
.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.

```csharp
using System;
using System.Data;
namespace TestScripts {
    public class Test {
        string _curUser = null;
        public void ModifyNextDownloadTimestamp (ref DateTime next_last_download, DateTime last_download, string user_name) {
            next_last_download = next_last_download.AddHours(-1);
        }
    }
}
```

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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>
### Parameter name for SQL scripts

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name. This is an INOUT parameter.</td>
<td>1</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

### 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 230.

### See also

- “Script parameters” on page 216
- “Script additions and deletions” on page 235
- “authenticate_user connection event” on page 266
- “authenticate_user_hashed connection event” on page 271
- “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.

class ExampleClass
{
    DBConnectionContext curConn;
    
    public ExampleClass( DBConnectionContext cc )
    {
        curConn = cc;
    }
    
    public void modifyUser( InOutString ioUserName )
        throws SQLException
    {
        Connection conn = curConn.getConnection();
        PreparedStatement uidSelect =
            conn.prepareStatement( "SELECT rep_id FROM SalesRep WHERE name = ?" );
        try {
            uidSelect.setString( 1, ioUserName.getValue() );
            ResultSet uidResult = uidSelect.executeQuery();
            try {
                if( uidResult.next() ) {
                    ioUserName.setValue( Integer.toString(uidResult.getInt( 1 )));
                }
            } finally {
                uidResult.close();
            }
        } finally {
            uidSelect.close();
        }
    }
}
.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.

```csharp
call ml_add_dnet_connection_script(
   'ver1',
   'modify_user',
   'TestScripts.Test.ModUser'
)
```

The following is the sample .NET method ModUser.

```csharp
using iAnywhere.MobiLink.Script;
namespace TestScripts
{
    public class Test
    {
        DBConnectionContext curConn;

        public Test( DBConnectionContext cc )
        {
            curConn = cc;
        }

        public void ModifyUser( ref string ioUserName )
        {
            DBCommand cmd = curConn.GetConnection().CreateCommand();
            cmd.CommandText = "SELECT rep_id FROM SalesRep WHERE name = ?";
            cmd.Parameters[0] = ioUserName;
            DBRowReader r = cmd.ExecuteReader();
            object[] row;
            if( (row = r.NextRow()) != null ) {
                ioUserName = (string) row[0];
            }
        }
    }
}
```

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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.
<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>1</td>
</tr>
<tr>
<td>s.last_download</td>
<td>TIMESTAMP. This is the last download time for the current synchronization.</td>
<td>2</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

**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.
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.

```
package ExamplePackage;
import ianywhere.ml.script.*;
import java.sql.*;
public class ExampleClass
{
    DBConnectionContext _cc;

    public ExampleClass( DBConnectionContext cc )
    {
        _cc = cc;
    }

    public void confirmDownload( String user,
                               Timestamp ts )
        throws SQLException
    {
        Connection conn = _cc.getConnection();
        PreparedStatement stmt = conn.prepareStatement(
            "INSERT INTO download_pubs_acked( rem_id, last_download ) " +
            "VALUES( ?,  ? )" );
        stmt.setString( 1, _cc.getRemoteID() );
        stmt.setTimestamp( 2, ts );
        stmt.executeUpdate();
        stmt.close();
    }
}
```

.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_download_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.

```csharp
using System;
using iAnywhere.MobiLink.Script;
namespace TestScripts
{
    public class Test
    {
        DBConnectionContext _cc;

        public Test(DBConnectionContext cc)
        {
            _cc = cc;
        }

        public void ConfirmDownload(string user, DateTime dt)
        {
            DBConnection conn = _cc.GetConnection();
            DBCommand cmd = conn.CreateCommand();
            cmd.CommandText = "INSERT INTO download_pubs_acked( rem_id, last_download ) VALUES( ?, ? )";
            cmd.Parameters[0] = _cc.GetRemoteID();
            cmd.Parameters[1] = dt;
            cmd.ExecuteNonQuery();
        }
    }
}
```

**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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.
<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.last_download</td>
<td>TIMESTAMP. The oldest download time of any synchronized table.</td>
<td>1</td>
</tr>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>2</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

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 216
- “Script additions and deletions” on page 235
- “end_upload connection event” on page 329
- “begin_download connection event” on page 277
- “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration]
- “Last download times in scripts” on page 100

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.

```sql
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.

```sql
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`.

```sql
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.

```java
public void prepareForDownload(
    Timestamp ts,
    String user ) {
    prepareT1ForDownload( _syncconn, user, ts );
    prepareT2ForDownload( _syncconn, user, ts );
}
```

**.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`.

```sql
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.

```csharp
public void PrepareForDownload(
    DateTime ts,
    string user ) {
    PrepareT1ForDownload ( _syncConn, user, ts );
    PrepareT2ForDownload ( _syncConn, user, ts );
}
```
**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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>1</td>
</tr>
<tr>
<td>s.last_publication_download</td>
<td>TIMESTAMP. The earliest last download time of any synchronized table.</td>
<td>2</td>
</tr>
<tr>
<td>s.publication name</td>
<td>VARCHAR(128). The name of the publication.</td>
<td>3</td>
</tr>
<tr>
<td>s.subscription_id</td>
<td>VARCHAR(128). The remote subscription ID.</td>
<td>4</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

**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 369
- dbmSync: “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.

```sql
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.

```sql
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.

```java
package ExamplePackage;
import anywhere.ml.script.*;
import java.sql.*;
public class ExampleClass
{
    public class ExampleClass()
    {
        DBConnectionContext _cc;
```
public ExampleClass( DBConnectionContext cc )
{
    _cc = cc;
}

public void confirmDownload( String user,
    Timestamp ts,
    String pubName )
throws SQLException
{
    Connection conn = _cc.getConnection();
    PreparedStatement stmt = conn.prepareStatement(
        "INSERT INTO download_pubs_acked( rem_id, last_download, pub_name )
        VALUES( ?,  ?, ? )" );
    stmt.setString( 1, _cc.getRemoteID() );
    stmt.setTimestamp( 2, ts );
    stmt.setString( 3, pubName );
    stmt.executeUpdate();
    stmt.close();
}

.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.

using System;
using iAnywhere.MobiLink.Script;
namespace TestScripts
{
    public class Test
    {
        DBConnectionContext _cc;

        public Test( DBConnectionContext cc )
        {
            _cc = cc;
        }

        public void ConfirmDownload( string user,
            DateTime dt,
            string pubName )
        {
            DBConnection conn = _cc.GetConnection();
            DBCommand cmd = conn.CreateCommand();
            cmd.CommandText =
                "INSERT INTO download_pubs_acked( rem_id, last_download, pub_name )
                VALUES( ?,  ?, ? )" ;
            cmd.Parameters[0] = _cc.GetRemoteID();
            cmd.Parameters[1] = dt;
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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.action_code</td>
<td>INTEGER. This is an INOUT parameter. This parameter is mandatory.</td>
<td>1</td>
</tr>
<tr>
<td>s.error_code</td>
<td>INTEGER. The native DBMS error code.</td>
<td>2</td>
</tr>
<tr>
<td>s.error_message</td>
<td>TEXT. The native DBMS error message.</td>
<td>3</td>
</tr>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>4</td>
</tr>
<tr>
<td>s.table</td>
<td>VARCHAR(128). The table whose script caused the error.</td>
<td>5</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>
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 or not 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 216
● “Script additions and deletions” on page 235
● “handle_error connection event” on page 344
● “handle_odbc_error connection event” on page 349
● “report_odbc_error connection event” on page 381
● “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.

```sql
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},
```

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0 379
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.

```java
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.

```java
package ExamplePackage;
import java.sql.*;
import ianywhere.ml.script.*;
public class ExampleClass
{
    DBConnectionContext _cc;

    public ExampleClass( DBConnectionContext cc )
    {
        _cc = cc;
    }

    public void reportError( ianywhere.ml.script.InOutInteger    actionCode, int errorCode, String errorMessage, String user, String table )
        throws SQLException
    {
        actionCode.setValue( errorCode );
        // Insert error information in a table,
        Connection conn = _cc.getConnection();
        PreparedStatement stmt = conn.prepareStatement(
            "INSERT INTO sync_error( action_code, error_code, error_message, user_name, table_name ) VALUES ( ?, ?, ?, ?, ? )" );
        stmt.setInt( 1, actionCode.getValue() );
        stmt.setInt( 2, errorCode);
        stmt.setString( 3, errorMessage);
        stmt.setString( 4, user);
        stmt.setString( 5, table);
        stmt.executeUpdate();
        stmt.close();
    }
}
```

.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.

```csharp
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.

```csharp
using System;
using iAnywhere.MobiLink.Script;
namespace TestScripts
{
    public class Test
    {
        DBConnectionContext _cc;

        public Test( DBConnectionContext cc )
        {
            _cc = cc;
        }

        public void ReportError( ref int actionCode,
            int errorCode,
            string errorMessage,
            string user,
            string table )
        {
            actionCode = errorCode;
            DBConnection conn = _cc.GetConnection();
            DBCommand cmd = conn.CreateCommand();
            cmd.CommandText = "INSERT INTO sync_error( action_code, error_code, error_message, " +
            "user_name, table_name ) VALUES ( ?, ?, ?, ?, ? )";
            cmd.Parameters[0] = actionCode;
            cmd.Parameters[3] = user;
            cmd.ExecuteNonQuery();
        }
    }
}
```

---

**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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going 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

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.action_code</td>
<td>INTEGER. This is an INOUT parameter. This parameter is mandatory.</td>
<td>1</td>
</tr>
<tr>
<td>s.odbc_state</td>
<td>VARCHAR(5). The ODBC SQLSTATE.</td>
<td>2</td>
</tr>
<tr>
<td>s.error_message</td>
<td>TEXT. The ODBC error message.</td>
<td>3</td>
</tr>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>4</td>
</tr>
<tr>
<td>s.table</td>
<td>VARCHAR(128). The table whose script caused the error.</td>
<td>5</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

#### 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 216
- “Script additions and deletions” on page 235
- “handle_error connection event” on page 344
- “handle_odbc_error connection event” on page 349
- “report_error connection event” on page 378
- “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.

```sql
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.

```java
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.

```java
package ExamplePackage;
import java.sql.*;
import ianywhere.ml.script.*;
public class ExampleClass
{
    DBConnectionContext _cc;

    public ExampleClass( DBConnectionContext cc )
    {
        _cc = cc;
    }

    public void reportODBCError( InOutInteger actionCode, 
                                  String odbcState, 
                                  String odbcMessage, 
                                  String user, 
                                  String table )
```
throws SQLException
{
    // Insert error information in a table,
    Connection conn = _cc.getConnection();
    PreparedStatement stmt = conn.prepareStatement(
        "INSERT INTO sync_error( action_code, odbc_state, error_message, " +
        "user_name, table_name ) VALUES ( ?, ?, ?, ?, ? )" );
    stmt.setInt( 1, actionCode.getValue() );
    stmt.setString( 2, odbcState );
    stmt.setString( 3, odbcMessage );
    stmt.setString( 4, user );
    stmt.setString( 5, table );
    stmt.executeUpdate();
    stmt.close();
}

.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.

    using System;
    using iAnywhere.MobiLink.Script;
    namespace TestScripts
    {
        public class Test
        {
            DBConnectionContext _cc;

            public Test( DBConnectionContext cc )
            {
                _cc = cc;
            }

            public void ReportODBCError( ref int actionCode,
                string odbcState,
                string errorMessage,
                string user,
                string table )
            {
                DBConnection conn = _cc.GetConnection();
                DBCommand cmd = conn.CreateCommand();
                cmd.CommandText =
                "INSERT INTO sync_error( action_code, odbc_state, error_message, " +
                "user_name, table_name ) VALUES ( ?, ?, ?, ?, ? )" ;
                cmd.Parameters[0] = actionCode;
                cmd.Parameters[1] = odbcState;
                cmd.Parameters[3] = user;
                cmd.ExecuteNonQuery();
            }
        }
    }
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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>1</td>
</tr>
<tr>
<td>s.table</td>
<td>VARCHAR(128). The table name.</td>
<td>2</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

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 216
- “Script additions and deletions” on page 235
- “upload_old_row_insert table event” on page 411
- “upload_new_row_insert table event” on page 409
- “upload_update table event” on page 422
- “end_upload_rows table event” on page 336
- “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.

```sql
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;
```

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.

```java
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.

```java
package ExamplePackage;
import java.sql.*;
import ianywhere.ml.script.*;
public class ExampleClass {
    DBConnectionContext _cc;

    public ExampleClass( DBConnectionContext cc ) {
        _cc = cc;
    }

    public void resolveConflict( String user, String table ) throws java.sql.SQLException {
        if( table == "Order" ) { // Insert error information in a table,
            Connection conn = _cc.getConnection();
            String conflictTable = "New" + table;
            PreparedStatement stmt = conn.prepareStatement("SELECT order_id, new_status, new_notes " + 
                    "FROM " + conflictTable + 
                    "WHERE rid = " + _cc.getRemoteID() );
            ResultSet rs = stmt.executeQuery();
            PreparedStatement updt = conn.prepareStatement( 
                    "UPDATE ULOrder SET status = ?, notes = ? WHERE order_id = ?" );
            while( rs.next() ) {
                if( rs.getString( 2 ) == "Approved" ) {
                    updt.setString( 1, rs.getString( 2 ) );
                    updt.setString( 2, rs.getString( 3 ) );
                    updt.setInt( 3, rs.getInt( 1 ) );
                }
            }
            updt.close();
            stmt.close();
        }
    }
}
```

.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.

```csharp
using System;
using iAnywhere.MobiLink.Script;
namespace TestScripts
{
    public class Test
    {
        DBConnectionContext _cc;

        public Test( DBConnectionContext cc )
        {
            _cc = cc;
        }

        public void ResolveConflict( string user,
                                      string table )
        {
            if( table == "Order" )
            {
                // Insert error information in a table,
                DBConnection conn = _cc.GetConnection();
                String conflictTable = "New" + table;
                DBCmd cmd = conn.CreateCommand();
                cmd.CommandText =
                        "SELECT order_id, new_status, new_notes " +
                        "FROM " + conflictTable +
                        "WHERE rid = " + _cc.GetRemoteID();
                DBRowReader dr = cmd.ExecuteReader();
                DBCmd updt = conn.CreateCommand();
                updt.CommandText =
                        "UPDATE ULOrder SET status = ?, notes = ? WHERE order_id = ?";
                object[] row;
                while( (row = dr.NextRow() ) != null )
                {
                    if( row[1].Equals( "Approved" ) )
                    {
                        updt.Parameters[0] = row[1];
                        updt.Parameters[1] = row[2];
                        updt.Parameters[2] = row[0];
                    }
                }
                updt.Close();
                cmd.Close();
                conn.Close();
            }
        }
    }
}
```

**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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>1</td>
</tr>
<tr>
<td>s.warnings</td>
<td>INTEGER. The number of warnings issued during the synchronization.</td>
<td>2</td>
</tr>
<tr>
<td>s.errors</td>
<td>INTEGER. The number of errors that occurred during the synchronization.</td>
<td>3</td>
</tr>
<tr>
<td>s.deadlocks</td>
<td>INTEGER. The number of deadlocks in the consolidated database that were detected for the synchronization.</td>
<td>4</td>
</tr>
<tr>
<td>s.synchronized_tables</td>
<td>INTEGER. The number of client tables that were involved in the synchronization.</td>
<td>5</td>
</tr>
<tr>
<td>s.connection_retries</td>
<td>INTEGER. The number of times the MobiLink server retried the connection to the consolidated database.</td>
<td>6</td>
</tr>
</tbody>
</table>
### Parameter name for SQL scripts

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

### 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 216
- “Script additions and deletions” on page 235
- “download_statistics connection event” on page 305
- “download_statistics table event” on page 308
- “upload_statistics connection event” on page 413
- “upload_statistics table event” on page 417
- “synchronization_statistics table event” on page 392
- “time_statistics connection event” on page 395
- “time_statistics table event” on page 398
- “MobiLink Profiler” on page 176
- “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.

```sql
CALL ml_add_connection_script('ver1',
  'synchronization_statistics',
  'INSERT INTO sync_con_audit(
    ml_user,
    warnings,
    errors,
```
deadlocks, synchronization_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. (Logging statistics to the MobiLink message log might be useful at development time but would slow down a production server.)

```java
package ExamplePackage;
public class ExampleClass {
  String _curUser = null;
  public void synchronizationStatisticsConnection(
    String user,
    int warnings,
    int errors,
    int deadlocks,
    int synchronizedTables,
    int connectionRetries ) {
    java.lang.System.out.println("synch statistics number of deadlocks: "+deadlocks);
  }
}
```

.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. (Logging statistics to the MobiLink message log might be useful at development time but would slow down a production server.)

```csharp
package ExamplePackage;
public class ExampleClass {
  String _curUser = null;
  public void synchronizationStatisticsConnection(
    String user,
    int warnings,
    int errors,
    int deadlocks,
    int synchronizedTables,
    int connectionRetries ) {
    java.lang.System.out.println("synch statistics number of deadlocks: "+deadlocks);
  }
}
namespace TestScripts
{
    public class Test
    {
        public void SyncStats( string user,
            int warnings,
            int errors,
            int deadLocks,
            int syncedTables,
            int connRetries )
        {
            System.Console.WriteLine( "synch statistics number of deadlocks: " + deadLocks );
        }
    }
}

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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>1</td>
</tr>
<tr>
<td>s.table</td>
<td>VARCHAR(128). The table name.</td>
<td>2</td>
</tr>
<tr>
<td>s.warnings</td>
<td>INTEGER. The number of warnings that occurred for the table during the synchronization.</td>
<td>3</td>
</tr>
</tbody>
</table>
### Parameter name for SQL scripts

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Description</th>
<th>Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.errors</td>
<td>INTEGER. The number of errors that were related to the table during the synchronization.</td>
<td>4</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

### 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 216
- “Script additions and deletions” on page 235
- “download_statistics connection event” on page 305
- “download_statistics table event” on page 308
- “upload_statistics connection event” on page 413
- “upload_statistics table event” on page 417
- “synchronization_statistics connection event” on page 388
- “time_statistics connection event” on page 395
- “time_statistics table event” on page 398
- “MobiLink Profiler” on page 176
- “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.

```sql
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. (Logging statistics to the MobiLink message log might be useful at development time but would slow down a production server.)

```
package ExamplePackage;
public class ExampleClass { 
    String _curUser = null;
    public void synchronizationStatisticsTable( 
        String user,
        String table, 
        int warnings, 
        int errors ) { 
        java.lang.System.out.println( "synch statistics for table: " + table + " warnings: " + warnings + " errors: " + errors ); 
    }
}
```

**.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. (Logging statistics to the MobiLink message log might be useful at development time but would slow down a production server.)

```
namespace TestScripts { 
    public class Test { 
        string _curUser = null;
        public void SyncTableStats( 
```
string user,
string table,
int warnings,
int errors ) {
    System.Console.WriteLine( "synch statistics for table: "
                        + table + " errors: " + errors );
}}}

**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, use the appropriate corresponding data type. See “SQL- Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>1</td>
</tr>
<tr>
<td>s.event_name</td>
<td>VARCHAR(128). The event whose statistics are being reported.</td>
<td>2</td>
</tr>
<tr>
<td>s.number_of_calls</td>
<td>INTEGER. The number of times the script was called.</td>
<td>3</td>
</tr>
<tr>
<td>s.minimum_time</td>
<td>INTEGER. Milliseconds. The shortest time it took to execute a script during this synchronization.</td>
<td>4</td>
</tr>
<tr>
<td>Parameter name for SQL scripts</td>
<td>Description</td>
<td>Order (deprecated for SQL)</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>s.maximum_time</td>
<td>INTEGER. Milliseconds. The longest time it took to execute a script during this synchronization.</td>
<td>5</td>
</tr>
<tr>
<td>s.total_time</td>
<td>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.)</td>
<td>6</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

**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 216
- “Script additions and deletions” on page 235
- “time_statistics table event” on page 398
- “download_statistics connection event” on page 305
- “download_statistics table event” on page 308
- “upload_statistics connection event” on page 413
- “upload_statistics table event” on page 417
- “synchronization_statistics connection event” on page 388
- “synchronization_statistics table event” on page 392
- “MobiLink Profiler” on page 176
- “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. (Printing statistics to the MobiLink message log might be useful at development time but would slow down a production server.)

package ExamplePackage;
public class ExampleClass
{
    public void timeStatisticsConnection(
        String    username,
        String    eventName,
        int        numberOfCalls,
        int        minimumTime,
        int        maximumTime,
        int        totalTime )
    {
        if( eventName.equals( "prepare_for_download") ) {
            System.out.println( "prepare_for_download num_calls: " +
                            numberOfCalls +
                            " total_time: " + totalTime );
        }
    }
}

.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',
    'ExamplePackage.ExampleClass.TimeStats' )

The following is the sample .NET method TimeStats. It prints statistics for the prepare_for_download event. (Printing statistics to the MobiLink message log might be useful at development time but would slow down a production server.)

namespace ExamplePackage
{
    public class ExampleClass
    {
        public void TimeStats(
            String    username,
            String    eventName,
            int        numberOfCalls,
            int        minimumTime,
            int        maximumTime,
            int        totalTime )
        {
            if( eventName.equals( "prepare_for_download") ) {
                System.out.println( "prepare_for_download num_calls: " +
                                    numberOfCalls +
                                    " total_time: " + totalTime );
            }
        }
    }
}
The following is the sample .NET method TimeStats. It prints statistics for the prepare_for_download event. (Printing statistics to the MobiLink message log might be useful at development time but would slow down a production server.)

```csharp	namespace TestScripts
{
    public class test
    {
        public void TimeStats( string user,
            string eventName,
            int numberOfCalls,
            int minimumTime,
            int maximumTime,
            int totTime )
        {
            if( eventName == "prepare_for_download" ) {
                System.Console.WriteLine( "prepare_for_download num_calls: " + numberOfCalls + "total_time: " + totTime );
            }
        }
    }
}
```

**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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Parameter name for SQL scripts</td>
<td>Description</td>
<td>Order (deprecated for SQL)</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>1</td>
</tr>
<tr>
<td>s.table</td>
<td>VARCHAR(128). The table name.</td>
<td>2</td>
</tr>
<tr>
<td>s.event_name</td>
<td>VARCHAR(128). The event whose statistics are being reported.</td>
<td>3</td>
</tr>
<tr>
<td>s.number_of_calls</td>
<td>INTEGER. The number of times the script was called.</td>
<td>4</td>
</tr>
<tr>
<td>s.minimum_time</td>
<td>INTEGER. Milliseconds. The shortest time it took to execute a script during the synchronization of this table.</td>
<td>5</td>
</tr>
<tr>
<td>s.maximum_time</td>
<td>INTEGER. Milliseconds. The longest time it took to execute a script during the synchronization of this table.</td>
<td>6</td>
</tr>
<tr>
<td>s.total_time</td>
<td>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.)</td>
<td>7</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

**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 216
- “Script additions and deletions” on page 235
- “time_statistics connection event” on page 395
- “download_statistics connection event” on page 305
- “download_statistics table event” on page 308
- “upload_statistics connection event” on page 413
- “upload_statistics table event” on page 417
- “synchronization_statistics connection event” on page 388
- “synchronization_statistics table event” on page 392
- “MobiLink Profiler” on page 176
- “Remote IDs and MobiLink user names in scripts” [MobiLink - Client Administration]

SQL example
The following example inserts statistical information into the time_statistics table.

```sql
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.

```java
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 void 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);
    }
}

.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 void TimeTableStats(
    string user,
    string table,
    string eventName,
    int numberOfCalls,
    int minimumTime,
    int maximumTime,
    int totalTime) {
    if (eventName == "upload_old_row_insert") {
        System.Console.WriteLine(
            "upload_old_row_insert num_calls: " + numCalls + 
            "total_time: " + total_time);
    }
}

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.
<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>N/A</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name. This parameter is optional.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>r.pk-column-1</td>
<td>Required. The first deleted primary key column value, referenced by column name or column number.</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>r.pk-column-N</td>
<td>Required. The last deleted primary key column value, referenced by column name or column number.</td>
<td>N</td>
</tr>
</tbody>
</table>

**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 585.

**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 422.
See also

- “Data scripts” on page 258
- “Script parameters” on page 216
- “Script additions and deletions” on page 235
- “upload_insert table event” on page 407
- “upload_update table event” on page 422

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.

```sql
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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink username. This parameter is optional.</td>
<td>Optional</td>
</tr>
</tbody>
</table>
### Parameter name for SQL scripts

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>r. primary-key-1</td>
<td>Required. The first primary key column value, referenced by column name or column number.</td>
<td>1 (2 if username is referenced)</td>
</tr>
<tr>
<td>r. primary-key-2</td>
<td>Required. The second primary key column value, referenced by column name or column number.</td>
<td>2</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>r. primary-key-N</td>
<td>Required. The last primary key column value, referenced by column name or column number.</td>
<td>N (N+1 if username is referenced)</td>
</tr>
</tbody>
</table>

### 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 585.

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 258
- “Script parameters” on page 216
- “Script additions and deletions” on page 235
- “Conflict detection” on page 116
- “resolve_conflict table event” on page 385
- “upload_delete table event” on page 401
- “upload_insert table event” on page 407
- “upload_update table event” on page 422
- "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 %SQLANYSAMP16% \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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>
### 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 422.

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

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>s.username</code></td>
<td>VARCHAR(128). The MobiLink username. This parameter is optional.</td>
<td>Optional</td>
</tr>
<tr>
<td><code>s.script_version</code></td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
<tr>
<td><code>r.pk-column-1</code></td>
<td>Required. The first primary key column value, referenced by column name or column number.</td>
<td>1 (2 if username is referenced)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td><code>r.pk-column-N</code></td>
<td>Required. The last primary key column value, referenced by column name or column number.</td>
<td>(N (N+1 \text{ if username is referenced}))</td>
</tr>
</tbody>
</table>
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 database are updated in the consolidated database.

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 585.

See also

- “Data scripts” on page 258
- “Script parameters” on page 216
- “Script additions and deletions” on page 235
- “Conflict detection” on page 116
- “upload_fetch table event” on page 403
- “resolve_conflict table event” on page 385
- “upload_delete table event” on page 401
- “upload_insert table event” on page 407
- “upload_update table event” on page 422

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Parameter name for SQL scripts</td>
<td>Description</td>
<td>Order (deprecated for SQL)</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name. This parameter is optional.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>r. column-1</td>
<td>Required. The first inserted column value, referenced by column name or column value.</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>r. column-N</td>
<td>Required. The last inserted column value, referenced by column name or column value.</td>
<td>N</td>
</tr>
</tbody>
</table>

**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 585.

**See also**

- “Data scripts” on page 258
- “Script parameters” on page 216
- “Script additions and deletions” on page 235
- “upload_delete table event” on page 401
- “upload_update table event” on page 422
- “upload_fetch table event” on page 403

**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.

```sql
CALL ml_add_table_script('ver1',
  '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 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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name. This parameter is optional.</td>
<td>Optional (1 if referenced)</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>r. column-1</td>
<td>Required. The first column value from the new (post-image) row, referenced by column name or column number.</td>
<td>1 (2 if username is referenced)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
### 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 422.

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 585.

#### See also

- “Data scripts” on page 258
- “Script parameters” on page 216
- “Script additions and deletions” on page 235
- “Conflict handling” on page 115
- “resolve_conflict table event” on page 385
- “upload_old_row_insert table event” on page 411
- “upload_update table event” on page 422
- “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 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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>N/A</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name. This parameter is optional.</td>
<td>Optional (1 if referenced)</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>r. column-1</td>
<td>Required. The first column value from the old (pre-image) row, referenced by column name or column number.</td>
<td>1 (2 if username is referenced)</td>
</tr>
</tbody>
</table>
**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 422.

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 585.

**See also**

- “Data scripts” on page 258
- “Script parameters” on page 216
- “Script additions and deletions” on page 235
- “Conflict handling” on page 115
- “resolve_conflict table event” on page 385
- “upload_new_row_insert table event” on page 409
- “upload_update table event” on page 422
- “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.

```sql
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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>Parameter name for SQL scripts</td>
<td>Description</td>
<td>Order (deprecated for SQL)</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink username.</td>
<td>1</td>
</tr>
<tr>
<td>s.warnings</td>
<td>INTEGER. The number of warnings that occurred.</td>
<td>2</td>
</tr>
<tr>
<td>s.errors</td>
<td>INTEGER. The number of errors that occurred.</td>
<td>3</td>
</tr>
<tr>
<td>s.inserted_rows</td>
<td>INTEGER. The number of rows that were successfully inserted in the consolidated database.</td>
<td>4</td>
</tr>
<tr>
<td>s.deleted_rows</td>
<td>INTEGER. The number of rows that were successfully deleted from the consolidated database.</td>
<td>5</td>
</tr>
<tr>
<td>s.updated_rows</td>
<td>INTEGER. The number of rows that were successfully updated in the consolidated database.</td>
<td>6</td>
</tr>
<tr>
<td>s.conflicted_updates</td>
<td>INTEGER. The number of update rows that caused conflict. A row is included only when a resolve conflict script was successfully called for it.</td>
<td>9</td>
</tr>
<tr>
<td>s.ignored_inserts</td>
<td>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 2) errors occurred when the MobiLink server was invoking the corresponding script and the handle_error or handle_odbc_error event returned 1000.</td>
<td>10</td>
</tr>
<tr>
<td>Parameter name for SQL scripts</td>
<td>Description</td>
<td>Order (deprecated for SQL)</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>s.ignored_deletes</td>
<td>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.</td>
<td>11</td>
</tr>
<tr>
<td>s.ignored_updates</td>
<td>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.</td>
<td>12</td>
</tr>
<tr>
<td>s.bytes</td>
<td>INTEGER. The amount of memory used within the MobiLink server to store the upload.</td>
<td>13</td>
</tr>
<tr>
<td>s.deadlocks</td>
<td>INTEGER. The number of deadlocks in the consolidated database that were detected for the synchronization.</td>
<td>14</td>
</tr>
</tbody>
</table>

**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 216
- “Script additions and deletions” on page 235
- “download_statistics connection event” on page 305
- “download_statistics table event” on page 308
- “upload_statistics table event” on page 417
- “synchronization_statistics connection event” on page 388
- “synchronization_statistics table event” on page 392
- “time_statistics connection event” on page 395
- “time_statistics table event” on page 398
- “MobiLink Profiler” on page 176
- “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`.

```sql
CALL ml_add_connection_script ( 'ver1', 'upload_statistics', 'INSERT INTO upload_summary_audit ( ml_user, warnings, errors, inserted_rows, deleted_rows, updated_rows, 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_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`.

```java
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. (Logging statistics to the MobiLink message log might be useful at development time but would slow down a production server.)

```java
package ExamplePackage;
public class ExampleClass { 
    String _curUser = null;
    public void 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 );
}

.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. (Logging statistics to the MobiLink message log might be useful at development time but would slow down a production server.)

namespace TestScripts {
    public class Test {
        string _curUser = null;
        public void 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 );
            }}

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, use the appropriate corresponding data type. See “SQL-Java data types” on page 430 and “SQL-.NET data types” on page 501.
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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter name for SQL scripts</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td>s.remote_id</td>
<td>VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
<tr>
<td>s.username</td>
<td>VARCHAR(128). The MobiLink user name.</td>
<td>1</td>
</tr>
<tr>
<td>s.table</td>
<td>VARCHAR(128). The table name.</td>
<td>2</td>
</tr>
<tr>
<td>s.warnings</td>
<td>INTEGER. The number of warnings issued in the upload of the table.</td>
<td>3</td>
</tr>
<tr>
<td>s.errors</td>
<td>INTEGER. The number of errors, including handled errors, that occurred in the upload of the table.</td>
<td>4</td>
</tr>
<tr>
<td>s.inserted_rows</td>
<td>INTEGER. The number of rows that were successfully inserted in the consolidated database.</td>
<td>5</td>
</tr>
<tr>
<td>s.deleted_rows</td>
<td>INTEGER. The number of rows that were successfully deleted from the consolidated database.</td>
<td>6</td>
</tr>
<tr>
<td>s.updated_rows</td>
<td>INTEGER. The number of rows that were successfully updated in the consolidated database.</td>
<td>7</td>
</tr>
<tr>
<td>Parameter name for SQL scripts</td>
<td>Description</td>
<td>Order (deprecated for SQL)</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>-------------</td>
<td>---------------------------</td>
</tr>
<tr>
<td>s.conflicted_updates</td>
<td>INTEGER. The number of update rows that caused conflict. A row is included only when a resolve conflict script was successfully called for it.</td>
<td>10</td>
</tr>
<tr>
<td>s.ignored_inserts</td>
<td>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 2) errors occurred when the MobiLink server was invoking the corresponding script and the handle_error or handle_odbc_error event returned 1000.</td>
<td>11</td>
</tr>
<tr>
<td>s.ignored_deletes</td>
<td>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.</td>
<td>12</td>
</tr>
<tr>
<td>s.ignored_updates</td>
<td>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.</td>
<td>13</td>
</tr>
<tr>
<td>s.bytes</td>
<td>INTEGER. The amount of memory used within the MobiLink server to store the upload.</td>
<td>14</td>
</tr>
<tr>
<td>s.deadlocks</td>
<td>INTEGER. The number of deadlocks in the consolidated database that were detected for the synchronization.</td>
<td>15</td>
</tr>
</tbody>
</table>

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 216
- “Script additions and deletions” on page 235
- “download_statistics connection event” on page 305
- “upload_statistics connection event” on page 413
- “upload_statistics table event” on page 417
- “synchronization_statistics connection event” on page 388
- “synchronization_statistics table event” on page 392
- “time_statistics connection event” on page 395
- “time_statistics table event” on page 398
- “MobiLink Profiler” on page 176
- “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_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_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_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_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. Logging statistics to the MobiLink message log might be useful at development time but would slow down a production server.

package ExamplePackage;
public class ExampleClass {
    String _curUser = null;
    public void 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 );
}

.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. (Logging statistics to the MobiLink message log might be useful at development time but would slow down a production server.)

namespace TestScripts {
public class Test {
    string _curUser = null;

public void 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 );
}}

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 are going to use parameter 2). If you use named parameters, you can specify any subset of the parameters in any order.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Order (deprecated for SQL)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>s. script_version</strong></td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
<tr>
<td><strong>r. column-1</strong></td>
<td>Required. The first non-primary key column value from the new (post-image) column value, referenced by column name or column number.</td>
<td>1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td><strong>r. column-M</strong></td>
<td>Required. The last non-primary key column value from the new (post-image) column value, referenced by column name or column number.</td>
<td>M</td>
</tr>
<tr>
<td><strong>r. pk-column-1</strong></td>
<td>Required. The first primary key column value from the new (post-image) column value, referenced by column name or column number.</td>
<td>M + 1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td><strong>r. pk-column-N</strong></td>
<td>Required. The last primary key column value from the new (post-image) column value, referenced by column name or column number.</td>
<td>M + N</td>
</tr>
<tr>
<td><strong>o. column-N</strong></td>
<td>Optional. The first non-primary key column value from the old (pre-image) column value, referenced by column name or column number.</td>
<td>M + N + 1</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td><strong>o. column-M</strong></td>
<td>Optional. The last non-primary key column value from the old (pre-image) column value, referenced by column name or column number.</td>
<td>M + N + M</td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Order (deprecated for SQL)</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------------------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>s.script_version</td>
<td>VARCHAR(128). Optional IN parameter to specify that the MobiLink server passes the script version string used for the current synchronization to this parameter. Question marks cannot be used to specify this parameter.</td>
<td>Not applicable</td>
</tr>
</tbody>
</table>

**Default action**

None.

**Remarks**

The WHERE clause must include all the primary key columns being synchronized and it can optionally include the non-primary key columns. 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_update 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 585.

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 database 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 585.
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.

```sql
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.

```sql
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.

Synchronization script writing 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

- “Options for writing server-side synchronization logic” [MobiLink - Getting Started]
- “Synchronization script writing” on page 211

Setting up Java synchronization logic

Use the following procedure to implement synchronization scripts in Java.

Prerequisites

There are no prerequisites for this task.

Context and remarks

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 %SQLANY16%\Java\mlscript.jar.

Task

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 431.

   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 431.

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 16\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 606
- “ml_add_java_connection_script system procedure” on page 612
- “ml_add_java_table_script system procedure” on page 614

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 mlsrv16 -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:

```
mlsrv16 -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 mlsrv16 option” on page 72.

- 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 mlsrv16 option” on page 72.

5. On Unix, 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.
Results

The Java synchronization logic is set up.

See also

- “Java synchronization logic” on page 429
- “Java synchronization example” on page 436
- “MobiLink server Java API reference” on page 440
- “Options for writing server-side synchronization logic” [MobiLink - Getting Started]
- “Synchronization script writing” on page 211

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 Java API reference” on page 440.

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 585.

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 502.

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.

<table>
<thead>
<tr>
<th>SQL data type</th>
<th>Corresponding Java data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>CHAR</td>
<td>java.lang.String</td>
</tr>
<tr>
<td>INTEGER</td>
<td>int or Integer</td>
</tr>
<tr>
<td>BINARY</td>
<td>byte[ ]</td>
</tr>
<tr>
<td>BIGINT</td>
<td>long</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>java.sql.Timestamp</td>
</tr>
<tr>
<td>INOUT INTEGER</td>
<td>ianywhere.ml.script.InOutInteger</td>
</tr>
<tr>
<td>INOUT VARCHAR</td>
<td>ianywhere.ml.script.InOutString</td>
</tr>
<tr>
<td>INOUT CHAR</td>
<td>ianywhere.ml.script.InOutString</td>
</tr>
<tr>
<td>INOUT BYTEARRAY</td>
<td>ianywhere.ml.script.InOutByteArray</td>
</tr>
<tr>
<td>INOUT TIMESTAMP</td>
<td>java.sql.Timestamp</td>
</tr>
</tbody>
</table>
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 \%SQLANY16\%java/mlscript.jar.

**Constructors**

The constructor of your class may have one of two possible signatures.

```java
public MyScriptClass(ianywhere.ml.script.DBConnectionContext sc)
```

or

```java
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 also**

- “DBConnectionContext interface [MobiLink server Java]” on page 440

**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 “Script additions and deletions” on page 235.

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 on the database upon its return, it can return null.

Java class debugging
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 16.

- 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 mlsrv16 command line option -verbose can be used.

  See “-v mlsrv16 option” on page 79.

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 mlsrv16 command line.

See “-sl java mlsrv16 option” on page 72.

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.

Note
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 about 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.

MobiLink server error handling 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.

```java
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
```
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

- “ServerContext.addErrorListener method [MobiLink server Java]” on page 469
- “ServerContext.removeErrorListener method [MobiLink server Java]” on page 474
- “ServerContext.addWarningListener method [MobiLink server Java]” on page 471
- “ServerContext.removeWarningListener method [MobiLink server Java]” on page 475
- “LogListener interface [MobiLink server Java]” on page 458
- “LogMessage class [MobiLink server Java]” on page 459

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 mlsrv16 -sl java option. For example, the following is part of a mlsrv16 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 mlsrv16 option” on page 72.

To see the start classes that are constructed at server start time, see “ServerContext.getStartClassInstances method [MobiLink server Java]” on page 473.
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.)

```java
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 {
        _sc = sc;
        _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.
        }
    }
}
```
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 427.
- 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.

```java
// 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.

```java
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 %sqlany16%\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:

mlsrv16 ... -sl java (-cp <class_location>)

MobiLink server Java API 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 %SQLANY16%\java\.

Package
ianywhere.ml.script

DBConnectionContext interface

Obtains information about the current database connection.

Syntax
public interface DBConnectionContext

Members
All members of the DBConnectionContext interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getConnection method</td>
<td>Returns the existing connection to the MobiLink consolidated database.</td>
</tr>
<tr>
<td>getDownloadData method</td>
<td>Returns the DownloadData for the current synchronization.</td>
</tr>
<tr>
<td>getNetworkData method</td>
<td>Returns information about the network streams for a synchronization.</td>
</tr>
<tr>
<td>getProperties method</td>
<td>Returns the Properties for this connection based on its script version.</td>
</tr>
<tr>
<td>getRemoteID method</td>
<td>Returns the remote ID of the database currently synchronizing on this connection.</td>
</tr>
</tbody>
</table>
### Description

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getServerContext method</code></td>
<td>Returns the ServerContext for this MobiLink server.</td>
</tr>
<tr>
<td><code>getVersion method</code></td>
<td>Returns the version string for this connection.</td>
</tr>
</tbody>
</table>

### 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 the `ServerContext` interface instead.

For more information about constructors, see “Constructors” on page 431.

### Note

A `DBConnectionContext` instance should not be used outside the thread that calls into your Java code.

### See also

- “ServerContext interface [MobiLink server Java]” on page 467

### 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.

```java
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.

### Syntax

```java
java.sql.Connection DBConnectionContext.getConnection()
    throws SQLException
```
Returns

An existing connection as a JDBC connection.

Exceptions

- **java.sql.SQLException**  Thrown when an error occurred binding the existing connection as a JDBC connection.

Remarks

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.

**Note**

Do not use the connection after the end_connection event has been called for that connection.

See also

- “ServerContext.makeConnection method [MobiLink server Java]” on page 473
- “DBConnectionContext interface [MobiLink server Java]” on page 440

**getDownloadData method**

Returns the DownloadData for the current synchronization.

**Syntax**

```java
DownloadData DBConnectionContext.getDownloadData()
```

**Returns**

DownloadData for the current synchronization. Null if this synchronization has no download.

**Remarks**

Use the DownloadData class to create the download for direct row handling. For more information about direct row handling, see “Direct row handling” on page 585.

See also

- “DownloadData interface [MobiLink server Java]” on page 445

**Example**

The following example shows you how to obtain a DownloadData instance for the current synchronization using the DBConnectionContext.getDownloadData method. This example assumes you have created a DBConnectionContext instance named _cc.

```java
// 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();
// ...
}
// ...

**getNetworkData method**

Returns information about the network streams for a synchronization.

**Syntax**

```
NetworkData DBConnectionContext.getNetworkData()
```

**Returns**

Information about the network streams used for the request, or null if the collection has not been enabled.

**Remarks**

This method is useful when authenticating against another server in the enterprise that uses the client-side certificate and HTTP headers.

To enable a collection of network stream data, add collect_network_data=1 to your -x switches. This option adds additional per-sync memory overhead to store the data.

**See also**

- “NetworkData interface [MobiLink server Java]” on page 463

**getProperties method**

Returns the Properties for this connection based on its script version.

**Syntax**

```
Properties DBConnectionContext.getProperties()
```

**Returns**

The properties for this connection.

**Remarks**

Properties are stored in the `ml_property` table. For more information, see “ml_add_property system procedure” on page 621.

Consult your Java Software Development Kit documentation for more information about `java.util.Properties`.

**Example**

The following example shows you how to output the properties for a `DBConnectionContext`. This example assumes you have a `DBConnectionContext` instance named `_cc`. 
getRemoteID method

Returns the remote ID of the database currently synchronizing on this connection.

Syntax

```java
String DBConnectionContext.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. This example assumes you have a DBConnectionContext instance named _cc.

```java
// 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

Returns the ServerContext for this MobiLink server.

Syntax

```java
ServerContext DBConnectionContext.getServerContext()
```

Returns

The MobiLink server context.

See also

● “ServerContext interface [MobiLink server Java]” on page 467

Example

The following example shows you how get the ServerContext instance for a DBConnectionContext. This example assumes you have a DBConnectionContext instance named _cc.

```java
// 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());
}
```
// A method that uses a ServerContext instance to shut down the server
public void shutDownServer() {
    ServerContext context = _cc.getServerContext();
    context.shutdown();
}

**getVersion method**

Returns the version string for this connection.

**Syntax**

```java
String DBConnectionContext.getVersion()
```

**Returns**

The script version.

**Remarks**

For more information, see “ml_add_property system procedure” on page 621.

**Example**

The following example shows you how to get the script version and base decisions on its value. This example assumes you have a `DBConnectionContext` instance named `_cc`.

```java
// 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")) {
        // ...
    }
    // ...
}
```

**DownloadData interface**

Encapsulates download data operations for direct row handling.

**Syntax**

```java
public interface DownloadData
```

**Members**

All members of the DownloadData interface, including all inherited members.
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 `DBConnectionContext.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_synchronization event or after the end_download event. It is not valid to access `DownloadData` in an upload-only synchronization. For more information, see “handle_DownloadData connection event” on page 341.

For more information about direct row handling, see “Direct row handling” on page 585.

See also

- “`DBConnectionContext.getDownloadData method [MobiLink server Java]`” on page 442
- “`DownloadTableData interface [MobiLink server Java]`” on page 448

Example

The following example shows you how to obtain a `DownloadData` instance for the current synchronization using the `DBConnectionContext.getDownloadData` method:

```java
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

Gets the named download table for this synchronization.

Syntax

```
DownloadTableData DownloadData.getDownloadTableByName(String table_name)
```
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.

Remarks

For more information about direct row handling, see “Direct row handling” on page 585.

See also

- “DownloadData interface [MobiLink server Java]” on page 445
- “DownloadTableData interface [MobiLink server Java]” on page 448
- “DBConnectionContext interface [MobiLink server Java]” on page 440

Example

The following example uses the `getDownloadTableByName` method to return a `DownloadTableData` instance for the `remoteOrders` table. This example assumes you have created a `DBConnectionContext` instance named `_cc`.

```java
// 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

Gets an array of all the tables for download in this synchronization.

Syntax

```java
DownloadTableData[] DownloadData.getDownloadTables()
```

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.

Remarks

The operations performed on this table are sent to the remote database.

For more information about direct row handling, see “Direct row handling” on page 585.
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 named `_cc`.

```java
// 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

Encapsulates information for one download table for a synchronization.

**Syntax**

```java
public interface DownloadTableData
```

**Members**

All members of the `DownloadTableData` interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getDeletePreparedStatement</code> method</td>
<td>Returns a <code>java.sql.PreparedStatement</code> instance that allows the user to add delete operations to the download.</td>
</tr>
<tr>
<td><code>getLastDownloadTime</code> method</td>
<td>Returns last download time for this table.</td>
</tr>
<tr>
<td><code>getMetaData</code> method</td>
<td>Gets the metadata for the <code>DownloadTableData</code> instance.</td>
</tr>
<tr>
<td><code>getName</code> method</td>
<td>Returns the table name for the <code>DownloadTableData</code> instance.</td>
</tr>
<tr>
<td><code>getUpsertPreparedStatement</code> method</td>
<td>Returns a <code>java.sql.PreparedStatement</code> instance which allows the user to add upsert (insert or update) operations to the download of a synchronization.</td>
</tr>
</tbody>
</table>
Remarks

Use this interface to set the data operations that are downloaded to the client.

You can use the DownloadData interface to obtain DownloadTableData instances for the current synchronization. You can use the getUpsertPreparedStatement and getDeletePreparedStatement methods to obtain Java prepared statements for insert and update, and delete operations, respectively.

You can execute delete statement with all primary keys set to null to have the remote client truncate the table.


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.

For more information about direct row handling, see “Direct row handling” on page 585.

See also

- “DownloadData interface [MobiLink server Java]” on page 445

Example

This example assumes that you have a table named remoteOrders in the MobiLink client databases that was created using the following SQL statement:

```sql
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. This example assumes you have a DBConnectionContext instance called _cc.

```java
// 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);
    // ...
}
```

See also

- “DownloadData interface [MobiLink server Java]” on page 445
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.

```java
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.

```java
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

Returns a `java.sql.PreparedStatement` instance that allows the user to add delete operations to the download.

**Syntax**

```java
java.sql.PreparedStatement DownloadTableData.getDeletePreparedStatement ()
    throws java.sql.SQLException
```

**Returns**

A `java.sql.PreparedStatement` instance for adding delete operations to the download.

**Exceptions**

- `java.sql.SQLException` Thrown if there is a problem retrieving the deleted `java.sql.PreparedStatement` instance.
Remarks

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.

<table>
<thead>
<tr>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>You must set all primary key values for download delete operations, or set all primary key values to null for truncate operations.</td>
</tr>
</tbody>
</table>

For more information about direct row handling, see “Direct row handling” on page 585.

See also

- “DownloadTableData interface [MobiLink server Java]” on page 448

Example

In the following example, the setDownloadDeletes method uses the 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.

```java
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();
}
```

getDownloadTime method

Returns last download time for this table.

Syntax

```
java.sql.Timestamp DownloadTableData.getLastDownloadTime()
```

Returns

The last download time for this download 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.

For more information about direct row handling, see “Direct row handling” on page 585.
See also

- “DownloadTableData interface [MobiLink server Java]” on page 448

Example

The following shows you how to populate a table in the download with inserts using the last download time. Note that this example assumes you have a \texttt{DBConnectionContext} instance called \_cc.

```java
// 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();

    // ...
}
```

\textbf{getMetaData method}

Gets the metadata for the \texttt{DownloadTableData} instance.

\textbf{Syntax}

```java
java.sql.ResultSetMetaData \texttt{DownloadTableData.getMetaData()}
```

\textbf{Returns}

The metadata for the \texttt{DownloadTableData} instance.

\textbf{Remarks}

The metadata is a standard \texttt{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.

For more information about direct row handling, see “Direct row handling” on page 585.

Consult your Java Software Development Kit documentation for more information about \texttt{java.sql.ResultSetMetaData}.

See also

- “DownloadTableData interface [MobiLink server Java]” on page 448
Example

The following example shows you how to get the number of columns used in the query for the DownloadTableData instance:

```java
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();
}
```

**getName method**

Returns the table name for the DownloadTableData instance.

**Syntax**

`String DownloadTableData.getName()`

**Returns**

The table name for the `DownloadTableData` instance.

**Remarks**

You can also access the table name using the `java.sql.ResultSetMetaData` instance returned by the `getMetaData` method.

For more information about direct row handling, see “Direct row handling” on page 585.

**See also**

- “DownloadTableData interface [MobiLink server Java]” on page 448
- “DownloadTableData.getMetaData method [MobiLink server Java]” on page 452

**Example**

The following example shows you how to output the table name for the `DownloadTableData` instance. This example assumes you have a `DBConnectionContext` instance named `_cc`.

```java
// 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);
```
getUpsertPreparedStatement method

Returns a java.sql.PreparedStatement instance which allows the user to add upsert (insert or update) operations to the download of a synchronization.

Syntax

```java
java.sql.PreparedStatement DownloadTableData.getUpsertPreparedStatement() throws java.sql.SQLException
```

Returns

A java.sql.PreparedStatement instance for adding upsert operations to the download.

Exceptions

- java.sql.SQLException Thrown if there is a problem retrieving the upserted java.sql.PreparedStatement instance.

Remarks

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.

For more information about direct row handling, see “Direct row handling” on page 585.

See also

- “DownloadTableData interface [MobiLink server Java]” on page 448

Example

In the following example, the setDownloadInserts method uses the 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.

```java
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();

### InOutInteger interface
Passed into methods to enable the functionality of an in/out parameter passed to a SQL script.

#### Syntax

```
public interface InOutInteger
```

#### Members

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getValue method</td>
<td>Returns the value of this integer parameter.</td>
</tr>
<tr>
<td>setValue method</td>
<td>Sets the value of this integer parameter.</td>
</tr>
</tbody>
</table>

#### Example

The following call to a MobiLink system procedure registers a Java method named `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 named `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);
    }
    // Insert was filtered because it was uploaded
    // in the same synchronization.
```
} 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**

Returns the value of this integer parameter.

**Syntax**

```java
int InOutInteger.getValue()
```

**Returns**

The value of this integer.

**See also**

- “InOutInteger interface [MobiLink server Java]” on page 455

**setValue method**

Sets the value of this integer parameter.

**Syntax**

```java
void InOutInteger.setValue(int new_value)
```

**Parameters**

- **new_value** The value this integer should take.

**See also**

- “InOutInteger interface [MobiLink server Java]” on page 455

**InOutString interface**

Passed into methods to enable the functionality of an in/out parameter passed to a SQL script.

**Syntax**

```java
public interface InOutString
```
Members

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getValue method</td>
<td>Returns the value of this String parameter.</td>
</tr>
<tr>
<td>setValue method</td>
<td>Sets the value of this String parameter.</td>
</tr>
</tbody>
</table>

Example

The following call to a MobiLink system procedure registers a Java method named modifyUser as the script for the modify_user connection event when synchronizing the script version ver1:

```java
CALL ml_add_java_connection_script(
    'ver1',
    'modify_user',
    'ExamplePackage.ExampleClass.modifyUser')
```

The following is the sample Java method named modifyUser. It gets the user ID from the database and then uses it to set the user name.

```java
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

Returns the value of this String parameter.

Syntax

```java
String InOutString.getValue()
```

Returns

The value of this String parameter.

See also

- “InOutString interface [MobiLink server Java]” on page 456
**setValue method**
Sets the value of this String parameter.

**Syntax**
```java
void InOutString.setValue(String new_value)
```

**Parameters**
- `new_value`  The value for this String to take.

**See also**
- “InOutString interface [MobiLink server Java]” on page 456

**LogListener interface**
Used for catching messages that are printed to the log.

**Syntax**
```java
public interface LogListener
```

**Members**
All members of the LogListener interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>messageLogged method</td>
<td>Invoked when a message is printed to the log.</td>
</tr>
</tbody>
</table>

**Remarks**
For more information, see “MobiLink server error handling in Java” on page 433.

**messageLogged method**
Invoked when a message is printed to the log.

**Syntax**
```java
void LogListener.messageLogged(ServerContext sc, LogMessage message)
```

**Parameters**
- `sc`  The context for the server that is printing the message.
- `message`  The LogMessage that has been sent to the MobiLink log.

**Remarks**
For more information, see “MobiLink server error handling in Java” on page 433.
LogMessage class

Holds the data associated with a log message.

Syntax

public class LogMessage

Members

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getText method</td>
<td>Accesses the text associated with this message.</td>
</tr>
<tr>
<td>getType method</td>
<td>Accesses this message type.</td>
</tr>
<tr>
<td>getUser method</td>
<td>Accesses the user name associated with this message.</td>
</tr>
<tr>
<td>ERROR variable</td>
<td>Indicates that the log message is an error.</td>
</tr>
<tr>
<td>INFO variable</td>
<td>Indicates that the message log displays information.</td>
</tr>
<tr>
<td>WARNING variable</td>
<td>Indicates that the log message is a warning.</td>
</tr>
</tbody>
</table>

Remarks

For more information, see “MobiLink server error handling in Java” on page 433.

ggetText method

Accesses the text associated with this message.

Syntax

String LogMessage.getText()

Returns

The main text of this message

See also

- “LogMessage.ERROR variable [MobiLink server Java]” on page 460
- “LogMessage.INFO variable [MobiLink server Java]” on page 461
- “LogMessage.WARNING variable [MobiLink server Java]” on page 462

ggetType method

Accesses this message type.
Syntax

```
int LogMessage.getType()
```

Returns
The type of this message, which can be either LogMessage.ERROR, LogMessage.INFO, or LogMessage.WARNING.

See also
- “LogMessage.ERROR variable [MobiLink server Java]” on page 460
- “LogMessage.INFO variable [MobiLink server Java]” on page 461
- “LogMessage.WARNING variable [MobiLink server Java]” on page 462

getUser method
Accesses the user name associated with this message.

Syntax

```
String LogMessage.getUser()
```

Returns
The user associated with this message. This value may be null if the message has no user.

See also
- “LogMessage.ERROR variable [MobiLink server Java]” on page 460
- “LogMessage.INFO variable [MobiLink server Java]” on page 461
- “LogMessage.WARNING variable [MobiLink server Java]” on page 462

ERROR variable
Indicates that the log message is an error.

Syntax

```
final int LogMessage.ERROR
```

See also
- “LogMessage class [MobiLink server Java]” on page 459

Example
The following code installs a LogListener object for all error messages and then writes the information to a file.

```
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 registers an ErrorLogListener object to receive error messages. 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.addErrorListener(new ErrorLogListener(outFile));

INFO variable

Indicates that the message log displays information.

Syntax

    final int LogMessage.INFO

See also

- “LogMessage class [MobiLink server Java]” on page 459
- “ServerContext.addInfoListener method [MobiLink server Java]” on page 469

Example

The following code installs a LogListener object for all information messages and then writes the information to a file.

    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 info output to the MobiLink log.
        e.printStackTrace();
    }
}

The following code registers an InfoLogListener object to receive info messages. 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.addInfoListener(new InfoLogListener(outFile));

**WARNING variable**

Indicates that the log message is a warning.

**Syntax**

    final int LogMessage.WARNING

**See also**

- “LogMessage class [MobiLink server Java]” on page 459

**Example**

The following code installs a LogListener object for all warning messages and then writes the information to a file.

    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 warning output to the MobiLink log.
        e.printStackTrace();
    }
}

The following code registers a WarningLogListener object 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;
    FileOutputStream outFile
    serv_context.addWarningListener(new WarningLogListener(outFile));

**NetworkData interface**

Contains information about the network streams for a synchronization.

**Syntax**

    public interface NetworkData

**Members**

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getCertificateChain method</td>
<td>Returns a java.security.cert.CertPath object containing any certificates sent by the client.</td>
</tr>
<tr>
<td>getHTTPHeaders method</td>
<td>Returns a Map object that maps header names to a List of header-values.</td>
</tr>
</tbody>
</table>
### Description

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getHTTPHeaderValue</code> method</td>
<td>Returns the value of the last header received by the server with the supplied name.</td>
</tr>
<tr>
<td><code>getHTTPHeaderValues</code> method</td>
<td>Returns all the header values received by the server associated with the supplied name.</td>
</tr>
<tr>
<td><code>isEndToEndEncrypted</code> method</td>
<td>Determines if this synchronization is end-to-end encrypted.</td>
</tr>
<tr>
<td><code>isHTTP</code> method</td>
<td>Determines if the synchronization uses HTTP or HTTPS.</td>
</tr>
<tr>
<td><code>isTLS</code> method</td>
<td>Determines if this synchronization uses TLS.</td>
</tr>
</tbody>
</table>

### Remarks

This interface is useful when authenticating against another server in the enterprise that uses the client-side certificate and HTTP headers.

To enable collection of network stream data, add `collect_network_data=1` to your `-x` switches. This adds additional per sync memory overhead to store the data. If using TLS or HTTPS with client-side certificates, add `trusted_certificates=<certificate file>=""` to have the server ask the client to send a certificate during the TLS handshake, incurring a time and network cost.

You can obtain a `NetworkData` object by invoking the `getNetworkData` method of the `DBConnectionContext` class. When using HTTP or HTTPS, it contains the header data for the last HTTP request received by the server before the authenticate scripts are invoked.

### See also

- “`DBConnectionContext interface [MobiLink server Java]`” on page 440

### Example

The following example illustrates how to get a `NetworkData` object from the `DBConnectionContext` object, and output the data.

```java
public class OrderProcessor {
    private DBConnectionContext _cc;

    public OrderProcessor( DBConnectionContext cc ) {
        _cc = cc;
    }

    // The method used for the authenticate_user event.
    public void AuthUser() {
        NetworkData nd = _cc.getNetworkData();
        if( nd != null ) {
            if( nd.isHTTP() ) {
                System.out.println( "http" );
                String user_agent = nd.getHTTPHeaderValue( "user-agent" );
                System.out.println( " user-agent: " +
                                    user_agent.substring( 0, user_agent.indexOf( '/' ) ) );
            } else {

```
System.out.println( "no http" );
}
if( nd.isTLS() ) {
    System.out.println( "tls" );
    CertPath certs = nd.getCertificateChain();
    if( certs != null ) {
        System.out.println( "  client-side cert:" );
        int n = 1;
        for( Certificate c : certs.getCertificates() ) {
            System.out.println( "    cert " + n++);
            X509Certificate c509 = (X509Certificate) c;
            System.out.println( "      Subject: " + c509.getSubjectX500Principal().getName() )
            System.out.println( "      Issuer: " + c509.getIssuerX500Principal().getName() );
        }
    } else {
        System.out.println( "  no client cert" );
    }
} else {
    System.out.println( "no tls" );
}
if( nd.isEndToEndEncrypted() ) {
    System.out.println( "e2ee" );
} else {
    System.out.println( "no e2ee" );
}
else {
    System.out.println( "NULL networkdata" );
}
}

getCertificateChain method

Returns a java.security.cert.CertPath object containing any certificates sent by the client.

Syntax

java.security.cert.CertPath NetworkData.getCertificateChain()

Returns

A CertPath containing the X509 certificates that identify the client; returns null if no such certificates were provided.

Remarks

The certificates are all java.security.cert.X509Certificate objects.

This method returns a non-null value only if the isTLS method returns true, the client supplies a certificate using the "identity" stream parameter, and the trusted_certificates option is set on the server. A non-null CertPath value contains the certificates, ordered from the self-signed certificate to the peer certificate.

See also

● “trusted_certificates example for Java” on page 92
getHTTPHeaders method

Returns a Map object that maps header names to a List of header-values.

Syntax

Map< String, List< String > > NetworkData.getHTTPHeaders()

Returns

A Map containing all the headers received by the server.

See also

- “NetworkData.getHTTPHeaderValue method [MobiLink server Java]” on page 466
- “NetworkData.getHTTPHeaderValues method [MobiLink server Java]” on page 466

getHTTPHeaderValue method

Returns the value of the last header received by the server with the supplied name.

Syntax

String NetworkData.getHTTPHeaderValue(String name)

Parameters

- name  The header name to return the value for.

Returns

The last header value associated with the supplied header name.

See also

- “NetworkData.getHTTPHeaderValues method [MobiLink server Java]” on page 466
- “NetworkData.getHTTPHeaders method [MobiLink server Java]” on page 466

getHTTPHeaderValues method

Returns all the header values received by the server associated with the supplied name.

Syntax

List< String > NetworkData.getHTTPHeaderValues(String name)

Parameters

- name  The header name to return the values for.

Returns

The header values associated with the supplied header name.
See also

- “NetworkData.getHTTPHeaderValue method [MobiLink server Java]” on page 466
- “NetworkData.getHTTPHeaders method [MobiLink server Java]” on page 466

**isEndToEndEncrypted method**

Determines if this synchronization is end-to-end encrypted.

**Syntax**

```java
boolean NetworkData.isEndToEndEncrypted()
```

**Returns**

True if this synchronization is end-to-end encrypted; otherwise, returns false.

**isHTTP method**

Determines if the synchronization uses HTTP or HTTPS.

**Syntax**

```java
boolean NetworkData.isHTTP()
```

**Returns**

True if this synchronization uses HTTP or HTTPS; otherwise, returns false.

**isTLS method**

Determines if this synchronization uses TLS.

**Syntax**

```java
boolean NetworkData.isTLS()
```

**Returns**

True if this synchronization uses TLS; otherwise, returns false.

**ServerContext interface**

An instantiation of all the context that is present for the duration of the MobiLink server.

**Syntax**

```java
public interface ServerContext
```

**Members**

All members of the ServerContext interface, including all inherited members.
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>addErrorListener method</td>
<td>Adds the specified LogListener object to receive a notification when an error is printed.</td>
</tr>
<tr>
<td>addInfoListener method</td>
<td>Adds the specified LogListener object from the list of Listener objects to receive a notification when info is printed.</td>
</tr>
<tr>
<td>addShutdownListener method</td>
<td>Adds the specified ShutdownListener object that is to receive notification before the server context is destroyed.</td>
</tr>
<tr>
<td>addWarningListener method</td>
<td>Adds the specified LogListener object to receive a notification when a warning is printed.</td>
</tr>
<tr>
<td>getProperties method</td>
<td>Returns the set of properties associated with a given component and property set.</td>
</tr>
<tr>
<td>getPropertiesByVersion method</td>
<td>Returns the set of properties associated with the script version.</td>
</tr>
<tr>
<td>getPropertySetNames method</td>
<td>Returns the list of property set names for a given component.</td>
</tr>
<tr>
<td>getStartClassInstances method</td>
<td>Gets an array of the start classes that were constructed at server start time.</td>
</tr>
<tr>
<td>makeConnection method</td>
<td>Opens and returns a new server connection.</td>
</tr>
<tr>
<td>removeErrorListener method</td>
<td>Removes the specified LogListener object from the list of Listener objects that receive notifications when an error is printed.</td>
</tr>
<tr>
<td>removeInfoListener method</td>
<td>Removes the specified LogListener object from the list of Listener objects that receive notifications when info is printed.</td>
</tr>
<tr>
<td>removeShutdownListener method</td>
<td>Removes the specified ShutdownListener object from the list of Listener objects that receive notifications before this ServerContext is destroyed.</td>
</tr>
<tr>
<td>removeWarningListener method</td>
<td>Removes the specified LogListener object from the list of Listener objects that receive notifications when a warning is printed.</td>
</tr>
<tr>
<td>shutdown method</td>
<td>Forces the server to shut down.</td>
</tr>
</tbody>
</table>

**Remarks**

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.
addErrorListener method

Adds the specified LogListener object to receive a notification when an error is printed.

Syntax

```java
void ServerContext.addErrorListener(LogListener ll)
```

Parameters

- `ll` The LogListener object to be notified on error.

Remarks

When an error is printed, the `LogListener.messageLogged(ServerContext, LogMessage)` method is called.

See also

- “LogListener.messageLogged method [MobiLink server Java]” on page 458
- “LogMessage class [MobiLink server Java]” on page 459

addInfoListener method

Adds the specified LogListener object from the list of Listener objects to receive a notification when info is printed.

Syntax

```java
void ServerContext.addInfoListener(LogListener ll)
```

Parameters

- `ll` The LogListener object to be notified on info.

Remarks

The `LogListener.messageLogged` method is called.

See also

- “LogListener.messageLogged method [MobiLink server Java]” on page 458

Example

The following code registers a `MyLogListener` object to receive notifications of info messages:

```java
// 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();
        }
    }
}

addShutdownListener method

Adds the specified ShutdownListener object that is to receive notification before the server context is destroyed.

Syntax

void ServerContext.addShutdownListener(ShutdownListener sl)

Parameters

● sl    The ShutdownListener object to be notified on shutdown.

Remarks

On shutdown, the ShutdownListener.shutdownPerformed(ServerContext) method is called.

See also

● “ShutdownListener interface [MobiLink server Java]” on page 477
● “ShutdownListener.shutdownPerformed method [MobiLink server Java]” on page 478
addWarningListener method

Adds the specified LogListener object to receive a notification when a warning is printed.

Syntax

```java
void ServerContext.addWarningListener(LogListener ll)
```

Parameters

- `ll` The LogListener object to be notified on warning.

Remarks

The LogListener.messageLogged(ServerContext, LogMessage) method is called.

See also

- “LogMessage class [MobiLink server Java]” on page 459
- “LogListener.messageLogged method [MobiLink server Java]” on page 458

getProperties method

Returns the set of properties associated with a given component and property set.

Syntax

```java
Properties ServerContext.getProperties(String component, String set)
```

Parameters

- `component` The component.
- `set` The property set.

Returns

The set of properties, which may be empty.

Remarks

These properties are stored in the ml_property system table. For more information, see “ml_add_property system procedure” on page 621.

Example

The following example shows how to lists all the properties on a ServerContext instance.

```java
import java.util.*;
ServerContext serverContext;
PrintStream out
Properties prop = serverContext.getProperties();
prop.list(out);
```
**getPropertiesByVersion method**

Returns the set of properties associated with the script version.

**Syntax**

```
Properties ServerContext.getPropertiesByVersion(String script_version)
```

**Parameters**

- `script_version` The script version for which to return the associated properties.

**Returns**

The set of properties associated with the given script version.

**Remarks**

These are stored in the `ml_property` system table. The script version is stored in the `property_set_name` column when the `component_name` is `ScriptVersion`. For more information, see “`ml_add_property` system procedure” on page 621.

**Example**

The following example shows how to lists all the properties on a `ServerContext` instance associated with a given script version:

```java
import java.util.*;
ServerContext serverContext;
PrintStream out
Properties prop = serverContext.getPropertiesByVersion("MyScriptVersion");
prop.list(out);
```

**getPropertySetNames method**

Returns the list of property set names for a given component.

**Syntax**

```
Iterator ServerContext.getPropertySetNames(String component)
```

**Parameters**

- `component` The name of the component for which to list property names.

**Returns**

The list of property set names for the given component.

**Remarks**

These properties are stored in the `ml_property` system table. For more information, see “`ml_add_property` system procedure” on page 621.
Example
The following example shows how to lists all the properties on a ServerContext instance associated with
a given component:

```java
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.

Syntax
```java
Object[] ServerContext.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.

Remarks
For more information, see “User-defined start classes” on page 434.

Example
The following example demonstrates how to use the getStartClassInstances method:

```java
Object objs[] = sc.getStartClassInstances();
int i;
for (i=0; i < objs.length; i += 1) {
    if (objs[i] instanceof MyClass) {
        // Use class.
    }
}
```

makeConnection method
Opens and returns a new server connection.

Syntax
```java
java.sql.Connection ServerContext.makeConnection() throws SQLException
```

Returns
The newly created server connection.

Exceptions
- `java.sql.SQLException` Thrown if an error occurred while opening a new connection.
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 the `DBConnectionContext.getServerContext` method on the `DBConnectionContext` for the current connection.

Note

Opening a connection can be expensive. Write your logic so that the number of calls to this method are minimized.

**removeErrorListener method**

Removes the specified `LogListener` object from the list of Listener objects that receive notifications when an error is printed.

**Syntax**

```java
void ServerContext.removeErrorListener(LogListener ll)
```

**Parameters**

- `ll` The Listener object to stop notifying.

**Example**

The following code removes a `LogListener` object from the list of error Listener objects:

```java
ServerContext serverContext;
LogListener myErrorListener;
serverContext.removeErrorListener(myErrorListener);
```

**removeInfoListener method**

Removes the specified `LogListener` object from the list of Listener objects that receive notifications when info is printed.

**Syntax**

```java
void ServerContext.removeInfoListener(LogListener ll)
```

**Parameters**

- `ll` The Listener object to stop notifying.

**Example**

The following code removes a `LogListener` object from the list of info Listener objects:

```java
ServerContext serverContext;
LogListener myInfoListener;
serverContext.removeInfoListener(myInfoListener);
```
removeShutdownListener method

Removes the specified ShutdownListener object from the list of Listener objects that receive notifications before this ServerContext is destroyed.

Syntax

```java
void ServerContext.removeShutdownListener(ShutdownListener sl)
```

Parameters

- `sl` The Listener object to stop notifying.

Example

The following code removes a ShutdownListener object from the list of Listener objects that are to receive notification before the ServerContext is destroyed:

```java
ServerContext serverContext;
ShutdownListener myShutdownListener
serverContext.removeShutdownListener(myShutdownListener);
```

removeWarningListener method

Removes the specified LogListener object from the list of Listener objects that receive notifications when a warning is printed.

Syntax

```java
void ServerContext.removeWarningListener(LogListener ll)
```

Parameters

- `ll` The Listener object to stop notifying.

Example

The following code removes a LogListener object from the list of warning Listener objects.

```java
ServerContext serverContext;
LogListener myWarningListener
serverContext.removeWarningListener(myWarningListener);
```

shutdown method

Forces the server to shut down.

Syntax

```java
void ServerContext.shutdown()
```

Remarks

Any registered ShutdownListener object have their shutdownPerformed method called.
**See also**

- “ShutdownListener.shutdownPerformed method [MobiLink server Java]” on page 478

**Example**

The following code forces the server to shut down:

```java
ServerContext serverContext;
serverContext.shutdown();
```

**ServerException class**

Thrown to indicate that there is an error condition that makes any further synchronization on the server impossible.

**Syntax**

```java
public class ServerException
```

**Members**

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServerException constructor</td>
<td>Constructs a ServerException.</td>
</tr>
</tbody>
</table>

**Remarks**

Throwing this exception causes the MobiLink server to shut down.

**Example**

The following example demonstrates how to throw a `ServerException` when a fatal problem occurs, and shut down the MobiLink server:

```java
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 constructor

Constructs a ServerException.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServerException() constructor</td>
<td>Constructs a ServerException with no detailed message.</td>
</tr>
<tr>
<td>ServerException(String) constructor</td>
<td>Constructs a ServerException with the specified detailed message.</td>
</tr>
</tbody>
</table>

ServerException() constructor

Constructs a ServerException with no detailed message.

Syntax

ServerException.ServerException()

ServerException(String) constructor

Constructs a ServerException with the specified detailed message.

Syntax

ServerException.ServerException(String s)

Parameters

- s A detailed message.

ShutdownListener interface

The Listener interface for catching server shutdowns.

Syntax

public interface ShutdownListener

Members

All members of the ShutdownListener interface, including all inherited members.
### shutdownPerformed method

Invoked before the **ServerContext** is destroyed due to server shutdown.

#### Syntax

```
void ShutdownListener.shutdownPerformed(ServerContext sc)
```

#### Parameters

- **sc**  
  The context for the server that is being shut down.

#### Remarks

Use this interface to ensure that all threads, connections, and other resources are cleaned up before the MobiLink server exits.

#### Example

The following example demonstrates how to install a `ShutdownListener` object for the **ServerContext** instance:

```java
class MyShutdownListener implements ShutdownListener {
    FileOutputStream _outFile;
    public MyShutdownListener(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 a `MyShutdownListener` object. Call this code from anywhere that has access to the **ServerContext** such as a class constructor or synchronization script.

```java
ServerContext serv_context;
FileOutputStream outFile;
serv_context.addShutdownListener(new MyShutdownListener(outFile));
```
SpatialUtilities class
A collection of static methods to work with spatial values.

Syntax
```java
public class SpatialUtilities
```

Members
All members of the SpatialUtilities class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>createSpatialValue</td>
<td>Returns a new byte array that contains a spatial value formatted for download.</td>
</tr>
<tr>
<td>getBytes method</td>
<td>Returns a new byte array with the same spatial data as the given byte array, but with the SRID removed.</td>
</tr>
<tr>
<td>getSRID method</td>
<td>Returns the SRID for the given spatial value.</td>
</tr>
<tr>
<td>setSRID method</td>
<td>Stores the given SRID in the first four bytes of the given byte array.</td>
</tr>
</tbody>
</table>

createSpatialValue method
Returns a new byte array that contains a spatial value formatted for download.

Syntax
```java
byte[] SpatialUtilities.createSpatialValue(
    int srid,
    byte[] spatial_value
)
```

Parameters
- `srid` The srid.
- `spatial_value` The spatial data.

Returns
The spatial value formatted for download.

Remarks
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.

Syntax

```java
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

```java
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

```java
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

Thrown to indicate that there is an error condition that makes the completion of the current synchronization impossible.
Syntax
public class SynchronizationException

Members
All members of the SynchronizationException class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SynchronizationException constructor</td>
<td>Constructs a SynchronizationException.</td>
</tr>
</tbody>
</table>

Remarks
Throwing this exception forces the MobiLink server to rollback the current synchronization and reinitialize the connection.

Example
The following example demonstrates how to throw a SynchronizationException if a problem occurs, which causes the MobiLink server to rollback.

```java
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());
        println("Inserts");
        printRSInfo(currentTable.getInserts());
        println("Updates");
        printUpdateRSInfo(currentTable.getUpdates());
        if (/* Reason for Sync failure */) {
            throw new SynchronizationException("Sync Failed");
        }
    }
}
```

SynchronizationException constructor
Constructs a SynchronizationException.
Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SynchronizationException() construc-</td>
<td>Constructs a <code>SynchronizationException</code> with no detailed message.</td>
</tr>
<tr>
<td>tor</td>
<td></td>
</tr>
<tr>
<td>SynchronizationException(String) constructor</td>
<td>Constructs a <code>SynchronizationException</code> with the specified detailed message.</td>
</tr>
</tbody>
</table>

**SynchronizationException() constructor**

Constructs a `SynchronizationException` with no detailed message.

Syntax

```
SynchronizationException.SynchronizationException()
```

**SynchronizationException(String) constructor**

Constructs a `SynchronizationException` with the specified detailed message.

Syntax

```
SynchronizationException.SynchronizationException(String s)
```

Parameters

- `s` A detailed 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 the `TimestampWithTimeZone` class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TimestampWithTimeZone constructor</td>
<td>Constructs a new <code>TimestampWithTimeZone</code> with the specified year, month, date, hour, minute, second, nano, time zone hour and time zone minute.</td>
</tr>
</tbody>
</table>
### Description

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>equals method</td>
<td>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.</td>
</tr>
<tr>
<td>getTimeZoneOffsetHours method</td>
<td>Gets the hours portion of the time zone offset.</td>
</tr>
<tr>
<td>getTimeZoneOffsetMinutes method</td>
<td>Gets the minutes portion of the time zone offset.</td>
</tr>
<tr>
<td>setTimeZoneOffsetHours method</td>
<td>Sets the hours portion of the time zone offset.</td>
</tr>
<tr>
<td>setTimeZoneOffsetMinutes method</td>
<td>Sets the minutes portion of the time zone offset.</td>
</tr>
<tr>
<td>toString method</td>
<td>Returns the string representing this timestamp with the form yyyy-mm-dd hh:mm:ss.SSSS Shh:mm, where S is the sign of the hours field.</td>
</tr>
<tr>
<td>toTimestampWithTimeZone method</td>
<td>Converts the given Timestamp to a TimestampWithTimeZone.</td>
</tr>
<tr>
<td>valueOf method</td>
<td>Converts a String to a TimestampWithTimeZone value.</td>
</tr>
</tbody>
</table>

### Remarks

Use this when using the MobiLink direct row API to specify the time zone offset for TIMESTAMP WITH TIME ZONE columns. PreparedStatement and ResultSet objects from JDBC drivers other than the MobiLink direct row API 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

```java
TimestampWithTimeZone.TimestampWithTimeZone (  
    int year,  
    int month,  
    int date,  
    int hour,  
    int minute,  
    int second,  
    int nano,  
    int tz_hour, 
```
Parameters

- **year**  The year minus 1900.
- **month**  An integer ranged from 0 to 11.
- **date**  An integer ranged from 1 to 31.
- **hour**  An integer ranged from 0 to 23.
- **minute**  An integer ranged from 0 to 59.
- **second**  An integer ranged from 0 to 59.
- **nano**  An integer ranged from 0 to 999,999,999.
- **tz_hour**  An integer ranged from -14 to 14.
- **tz_minute**  An integer ranged from 0 to 59.

Exceptions

- **java.lang.IllegalArgumentException**  Thrown if tz_minute is not in 0 - 59 or tz_hour is not within the appropriate range.

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>equals(Object) method</td>
<td>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.</td>
</tr>
<tr>
<td>equals(Timestamp) method</td>
<td>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.</td>
</tr>
</tbody>
</table>

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)

Parameters
   - o  The object to compare against.

Returns
   True if o is equal to this; otherwise false.

See also
   - “TimestampWithTimeZone.equals method [MobiLink server Java]” on page 484

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.

See also
   - “TimestampWithTimeZone.equals method [MobiLink server Java]” on page 484

g getTimeZoneOffsetHours method
   Gets the hours portion of the time zone offset.

Syntax
   int TimestampWithTimeZone.getTimeZoneOffsetHours ()

Returns
   The hours portion of the time zone offset.

g 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  Thrown 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  Thrown if tz_minute is not in 0 - 59.

**toString method**

Returns the string representing this timestamp with the form *yyyyMMdd 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

```java
TimestampWithTimeZone toTimestampWithTimeZone(Timestamp ts)
```

Parameters

- `ts` The timestamp to convert.

Returns

- `ts` if `ts` is an instance of `TimestampWithTimeZone`. Otherwise, constructs a new `TimestampWithTimeZone` that is equivalent to `ts` with a time zone offset of 00:00.

**valueOf method**

Converts a String to a `TimestampWithTimeZone` value.

Syntax

```java
valueOf(String val)
```

Parameters

- `val` A timestamp with time zone in the `yyyy-mm-dd hh:mm:ss[.f...][+-][hh:mm]` format. The fractional and time zone parts may be omitted. If the time zone is present, the sign can be omitted.

Returns

- The new `TimestampWithTimeZone`.

Exceptions

- `java.lang.IllegalArgumentException` Thrown if `val` does not have the correct format.

**UpdateResultSet interface**

A result set object that includes special methods for accessing the pre-image (old) and post-image (new) values of a specified row.

Syntax

```java
public interface UpdateResultSet
```

Members

- All members of the `UpdateResultSet` interface, including all inherited members.
### Name and Description

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>setNewRowValues</code></td>
<td>Sets the mode of this result set to return new column values (the post update row).</td>
</tr>
<tr>
<td><code>setOldRowValues</code></td>
<td>Sets the mode of this result set to return old column values (the pre update row).</td>
</tr>
</tbody>
</table>

### Remarks

You can hold the update operations for one upload transaction for one table.

New and old rows can both be accessed by changing the mode of the `ResultSet` to old or new.

Use the `DownloadTableData.getUpdates` method to obtain an `UpdateResultSet` instance.

`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`.

For more information about handling direct upload conflicts, see “Direct upload conflicts” on page 589.

For more information about direct row handling, see “Direct row handling” on page 585.

### setNewRowValues method

Sets the mode of this result set to return new column values (the post update row).

#### Syntax

```java
void UpdateResultSet.setNewRowValues()
```

#### Remarks

The result set represents the latest updated values in the remote client database.

This is the default mode.

For more information about handling direct upload conflicts, see “Direct upload conflicts” on page 589.

For more information about direct row handling, see “Direct row handling” on page 585.

#### Example

The following code shows how to set the mode of the `UpdateResultSet` to return new column values:
setOldRowValues method

Sets the mode of this result set to return old column values (the pre update row).

Syntax

```java
void UpdateResultSet.setOldRowValues()
```

Remarks

In this mode, the `UpdateResultSet` represents old column values obtained by the client in the last synchronization.

For more information about handling direct upload conflicts, see “Direct upload conflicts” on page 589.

For more information about direct row handling, see “Direct row handling” on page 585.

Example

The following code shows how to set the mode of the `UpdateResultSet` to return old column values:

```java
UpdateResultSet results
results.setOldRowValues();
```

UploadData interface

Encapsulates upload operations for direct row handling.

Syntax

```java
public interface UploadData
```

Members

All members of the `UploadData` interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getUploadedTableByName method</td>
<td>Returns an <code>UploadedTableData</code> instance representing the specified table.</td>
</tr>
<tr>
<td>getUploadedTables method</td>
<td>Returns an array of <code>UploadedTableData</code> objects for the current synchronization.</td>
</tr>
</tbody>
</table>

Remarks

An `UploadData` instance representing a single upload transaction is passed to the handle_UploadData 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 getUploadedTables or getUploadedTableByName methods to obtain UploadedTableData instances.

A synchronization has one UploadData unless the remote database is using transactional upload.

For more information about direct row handling, see “Direct row handling” on page 585.

For more information about handling direct upload conflicts, see “Direct upload conflicts” on page 589.

For more information about the handle_UploadData connection event, see “handle_UploadData connection event” on page 352.

See also
● “UploadedTableData interface [MobiLink server Java]” on page 491

getUploadedTableByName method

Returns an UploadedTableData instance representing the specified table.

Syntax

UploadedTableData UploadData.getUploadedTableByName(String table_name)

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

● “UploadedTableData interface [MobiLink server Java]” on page 491

Example

Assume you use a method named HandleUpload for the handle_UploadData event. The following example uses the GetUploadedTableByName method to return an UploadedTableData instance for the remoteOrderstable.

```java
public void handleUpload(UploadData ud)
    throws SQLException, IOException
{
    UploadedTableData uploaded_t1 =
    ud.GetUploadedTableByName("remoteOrders");
}
getUploadedTables method

Returns an array of UploadedTableData objects for the current synchronization.

Syntax

```java
UploadedTableData[] UploadData.getUploadedTables()
```

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.

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. If your data source is a relational database, use this table order.

See also

- “UploadedTableData interface [MobiLink server Java]” on page 491

Example

Assume you use a method named HandleUpload for the handle_UploadData event. The following example uses the getUploadedTables method to return UploadedTableData instances for the current upload transaction.

```java
public void handleUpload(UploadData ud)
    throws SQLException, IOException
{
    UploadedTableData tables[] = ud.getUploadedTables();
    //...
}
```

UploadedTableData interface

Encapsulates table operations for direct row handling uploads.

Syntax

```java
public interface UploadedTableData
```

Members

All members of the UploadedTableData interface, including all inherited members.
### Name | Description
--- | ---
getDeletes method | Returns a java.sql.ResultSet object representing delete operations uploaded by a MobiLink client.
getInserts method | Returns a java.sql.ResultSet object representing insert operations uploaded by a MobiLink client.
getMetaData method | Gets the metadata for the UploadedTableData instance.
getName method | Returns the table name for the UploadedTableData instance.
getUpdates method | Returns a UpdateResultSet object representing update operations uploaded by a MobiLink client.

**Remarks**

You can use an UploadedTableData instance to obtain a table's insert, update, and delete operations for a single upload transaction. Use the getInserts, getUpdates, and 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 getMetaData method or the result sets returned by getInserts, getUpdates, and getDeletes. The delete result set only includes primary key columns for a table.

For more information about direct row handling, see “Direct row handling” on page 585.

For more information about the handle_UploadData connection event, see “handle_UploadData connection event” on page 352.

**See also**

- “UploadData interface [MobiLink server Java]” on page 489

**Example**

The following code gets the deletes uploaded and prints out the first column of each:

```java
void printFirstColOfDeletes( UploadedTableData tab_data )
{
    ResultSet deletes = tab_data.getDeletes();
    while( deletes.next() ) {
        java.lang.System.out.println( deletes.getString( 1 ) );
    }
    deletes.close();
}
```

The following code prints the new and old value of the third column of each update:

```java
void printThirdColOfUpdates( UploadedTableData tab_data )
{
    ResultSet updates = tab_data.getUpdates();
```
while( updates.next() ) {
    updates.setOldRowValues();
    java.lang.System.out.println( "old row col: " +
    updates.getString( 3 ) );
    updates.setNewRowValues();
    java.lang.System.out.println( "new row col: " +
    updates.getString( 3 ) );
}  
updates.close();

getDeletes method

Returns a java.sql.ResultSet object representing delete operations uploaded by a MobiLink client.

Syntax

java.sql.ResultSet UploadedTableData.getDeletes()

Returns

A java.sql.ResultSet object that represents delete operations uploaded by a MobiLink client.

Remarks

The result set contains primary key values for deleted rows.

For more information about direct row handling, see “Direct row handling” on page 585.

For more information about the handle_UploadData connection event, see “handle_UploadData connection event” on page 352.

Example

Assume your remote client contains a table named 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);
        // ...
    
}
getInserts method

Returns a `java.sql.ResultSet` object representing insert operations uploaded by a MobiLink client.

Syntax

```java
java.sql.ResultSet UploadedTableData.getInserts()
```

Returns

A `java.sql.ResultSet` object representing insert operations uploaded by a MobiLink client.

Remarks

Each Insert is represented by one row in the result set.

For more information about direct row handling, see “Direct row handling” on page 585.

Example

Assume your remote client contains a table named `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.

```java
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();
}
```

getMetaData method

Gets the metadata for the `UploadedTableData` instance.

Syntax

```java
java.sql.ResultSetMetaData UploadedTableData.getMetaData()
```
Returns
The metadata for the UploadedTableData instance.

Remarks
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.

For more information about direct row handling, see “Direct row handling” on page 585.

For more information about the handle.UploadData connection event, see “handle.UploadData connection event” on page 352.

Example
The following example obtains a java.sql.ResultSetMetaData instance for an uploaded table named remoteOrders. The code uses the ResultSetMetaData.getColumnCount and getColumnLabel methods to compile a list of column names.

```java
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 += ", ";
                   }
               }

               //...

               // Get a String list of column names.
               }

In this case, a method named HandleUpload handles the handle.UploadData synchronization event.
**getName method**

Returns the table name for the UploadedTableData instance.

**Syntax**

```java
String UploadedTableData.getName()
```

**Returns**

The table name for the UploadedTableData instance.

**Remarks**

You can also access the table name using the `java.sql.ResultSetMetaData` instance returned by the `getMetaData` method.

For more information about direct row handling, see “Direct row handling” on page 585.

For more information about the handle_UploadData connection event, see “handle_UploadData connection event” on page 352.

**See also**

- “UploadedTableData.getMetaData method [MobiLink server Java]” on page 494

**Example**

The following example obtains the name of each uploaded table in a single upload transaction:

```java
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++) {
        // Get the table name.
        String table_name = tables[i].getName();
        // ...
    }
}
```

**getUpdates method**

Returns a UpdateResultSet object representing update operations uploaded by a MobiLink client.

**Syntax**

```java
UpdateResultSet UploadedTableData.getUpdates()
```
Returns
An UpdateResultSet object representing update operations uploaded by a MobiLink client.

Remarks
Each update is represented by one row including all column values. UpdateResultSet extends java.sql.ResultSet to include special methods for MobiLink conflict detection.

For more information about direct row handling, see “Direct row handling” on page 585.

For more information about handling direct upload conflicts, see “Direct upload conflicts” on page 589.

For more information about the handle_UploadData connection event, see “handle_UploadData connection event” on page 352.

See also
- “UpdateResultSet interface [MobiLink server Java]” on page 487

Example
Assume your remote client contains a table named remoteOrders. The following example uses the 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();
}

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
- “Options for writing server-side synchronization logic” [MobiLink - Getting Started]
- “Synchronization script writing” on page 211

Implementing synchronization scripts in .NET

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.

Prerequisites

There are no prerequisites for this task.

Task

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 502.

   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 502.

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 %SQLANY16%\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” on page 512.

3. Compile your project.

   For example, compile from Visual Studio as follows:
a. On the VS.NET Project menu, click Add Existing Item.

   In the Open list, click 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#:

```
```

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 610 and “ml_add_dnet_table_script system procedure” on page 611.

You can also add this information using Sybase Central.

See “Script additions and deletions” on page 235.

5. Instruct the MobiLink server to load assemblies and start the CLR. You tell MobiLink where to locate these assemblies using options in the mlsrv16 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:

```
mlsrv16 -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 “.NET assembly loading” on page 508 and “-sl dnet mlsrv16 option” on page 71.

• **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 “.NET assembly loading” on page 508.

For more information about the mlsrv16 option -sl dnet, see “-sl dnet mlsrv16 option” on page 71.

**Note**

You can use the -MLAutoLoadPath option or the -MLDomConfigFile option, but not both.

**Results**

The .NET synchronization logic is set up.

**.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” on page 512.

.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 585.
**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 502.

**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.

<table>
<thead>
<tr>
<th>SQL data type</th>
<th>Corresponding .NET data type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR</td>
<td>string</td>
</tr>
<tr>
<td>CHAR</td>
<td>string</td>
</tr>
<tr>
<td>INTEGER</td>
<td>int</td>
</tr>
<tr>
<td>BIGINT</td>
<td>long</td>
</tr>
<tr>
<td>SQL data type</td>
<td>Corresponding .NET data type</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>BINARY</td>
<td>byte [ ]</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>DateTime</td>
</tr>
<tr>
<td>INOUT INTEGER</td>
<td>ref int</td>
</tr>
<tr>
<td>INOUT VARCHAR</td>
<td>ref string</td>
</tr>
<tr>
<td>INOUT CHAR</td>
<td>ref string</td>
</tr>
<tr>
<td>INOUT BYTEARRAY</td>
<td>ref byte [ ]</td>
</tr>
<tr>
<td>INOUT TIMESTAMP</td>
<td>ref DateTime</td>
</tr>
</tbody>
</table>

**Constructors**

The constructor of your class takes no parameters or takes one `iAnywhere.MobiLink.Script.DBConnectionContext` parameter. For example:

```csharp
public ExampleClass(iAnywhere.MobiLink.Script.DBConnectionContext cc)
```

or

```csharp
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 an `iAnywhere.MobiLink.Script.DBConnectionContext` parameter if it exists. If it does not, it uses the void constructor.

See “`DBConnectionContext interface [MobiLink server .NET]`” on page 524.

**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 on 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 mlsrv16 -sl dnet option. For example, the following is part of an mlsrv16 command line. It causes mycl1 and mycl2 to be loaded as start classes.

```
-sldnet(-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 mlsrv16 option” on page 71.

To see the start classes that are constructed at server start time, see “ServerContext.GetStartClassInstances method [MobiLink server .NET]” on page 564.

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) {
            _sc = sc;
            _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);
    }
}

How to print 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.

Note
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.

MobiLink server error handling 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;
        }
    }
}
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.

```csharp
// ServerContext serv_context;
TestLogListener errtll = new TestLogListener(log_listener_file);
serv_context.ErrorListener += new LogCallback(errtll.errCallback);
```

See also
- “LogCallback delegate [MobiLink server .NET]” on page 581
- “ServerContext.ErrorListener event [MobiLink server .NET]” on page 565
- “ServerContext.WarningListener event [MobiLink server .NET]” on page 566
- “LogMessage class [MobiLink server .NET]” on page 553
- “LogMessage.MessageType enumeration [MobiLink server .NET]” on page 554

## Debugging .NET synchronization logic

The following procedure can be used to debug your .NET scripts using Visual Studio.

### Prerequisites

There are no prerequisites for this task.

### Task

2. Select **Tools » Attach to Process**.
3. In the **Available Processes** control, select **mlsrv16.exe** and then press **Attach**.
4. Set your break points.
5. Start a synchronization.

### Results

The script can be debugged.

The following procedure can be used to debug your .NET scripts using Visual Studio.

### Prerequisites

There are no prerequisites for this task.
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.
     - Click Build » Configuration Manager.
       - In the Active Solution Configuration list, click 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 mlsrv16 options, including the connection string and the option to load .NET assemblies.
     For 32-bit Windows environments:
     
     \devenv /debugexe %sqlany16%\bin32\mlsrv16.exe -c ... 
     
     For 64-bit Windows environments:
     
     \devenv /debugexe %sqlany16%\bin64\mlsrv16.exe -c ... 
     
     Visual Studio starts and mlsrv16.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 mlsrv16.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. In Visual Studio 2010, change Debugger Type to Managed(v2.0, v1.1, v1.0) or Managed v4.0, depending on the assembly version used by the MobiLink server.

   **Note**
   To use the v4.0 assemblies, you must explicitly include the -clrVersion option when you load the MobiLink server. For more information about the -clrVersion option, see “-sl dnet mlsrv16 option” on page 71.

5. Open the associated .NET source files and set break points.
   - Open the source files individually in the mlsrv16 solution. Do not open the original solution or project file.
6. Start MobiLink from the Debug menu or by pressing F5.

   If prompted, save mlsrv16.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.

Results

The script can be debugged.

.NET synchronization techniques

This section describes techniques you can use to tackle common .NET synchronization tasks.

Row uploads and downloads

For information about how to upload or download rows via .NET, see “Direct row handling” on page 585.

How to load shared assemblies

This section details options to load .NET assemblies and details the process to load shared assemblies.

.NET assembly loading

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
<?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>
    </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 mlsrv16 option `-sl dnet`, see “-sl dnet mlsrv16 option” on page 71.

### .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.

1. Add the table user_pwd_table to the database. Execute the following SQL statements in Interactive SQL:

   ```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:

   ```sql
   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 266.

   ```csharp
   using System;
   using iAnywhere.MobiLink.Script;
   namespace MLEexample {

   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;
       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:"%SQLANY16%\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**

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 %SQLANY16%\Assembly\V2.

**Namespace**

iAnywhere.MobiLink.Script

**DateTimeWithTimeZone class**

Represents a DateTime with time zone offsets.

**Visual Basic syntax**

```vbnet
Public NotInheritable Class DateTimeWithTimeZone
```

**C# syntax**

```csharp
public sealed class DateTimeWithTimeZone
```
Members

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DateTimeWithTimeZone const-</td>
<td>Constructs a DateTimeWithTimeZone with the specified year, month, day, hour, minute, second, millisecond, time zone hour and time zone minute.</td>
</tr>
<tr>
<td>ructor</td>
<td></td>
</tr>
<tr>
<td>Parse method</td>
<td>Parses the given string and returns a new DateTimeWithTimeZone.</td>
</tr>
<tr>
<td>ToString method</td>
<td>The equivalent to DateTime.ToString() with the time zone offset appended to the end.</td>
</tr>
<tr>
<td>DateTime property</td>
<td>Gets the DateTime that corresponds to this without the time zone offset.</td>
</tr>
<tr>
<td>Day property</td>
<td>The equivalent to DateTime.Day.</td>
</tr>
<tr>
<td>Hour property</td>
<td>The equivalent to DateTime.Hour.</td>
</tr>
<tr>
<td>Millisecond property</td>
<td>The equivalent to DateTime.Millisecond.</td>
</tr>
<tr>
<td>Minute property</td>
<td>The equivalent to DateTime.Minute.</td>
</tr>
<tr>
<td>Month property</td>
<td>The equivalent to DateTime.Month.</td>
</tr>
<tr>
<td>Second property</td>
<td>The equivalent to DateTime.Second.</td>
</tr>
<tr>
<td>TimeZoneHour property</td>
<td>Gets the hours part of the time zone offset.</td>
</tr>
<tr>
<td>TimeZoneMinute property</td>
<td>Gets the minutes part of the time zone offset.</td>
</tr>
<tr>
<td>Year property</td>
<td>The equivalent to DateTime.Year.</td>
</tr>
</tbody>
</table>

DateTimeWithTimeZone constructor

Constructs a DateTimeWithTimeZone with the specified year, month, day, hour, minute, second, millisecond, time zone hour and time zone minute.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DateTimeWithTimeZone(DateTime) constructor</td>
<td>Constructs a DateTimeWithTimeZone with the same date and time as the specified DateTime with time zone offsets of 0.</td>
</tr>
</tbody>
</table>
**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**

```vbnet
Public Sub New(ByVal dt As Date)
```

**C# syntax**

```csharp
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**

```vbnet
Public Sub New(ByVal dt As Date, ByVal tz_hour As Integer, ByVal tz_minute As Integer)
```

**C# syntax**

```csharp
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) 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.fffffff 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToString() method</td>
<td>The equivalent to DateTime.ToString() with the time zone offset appended to the end.</td>
</tr>
<tr>
<td>ToString(IFormatProvider) method</td>
<td>The equivalent to DateTime.ToString(provider) with the time zone offset appended to the end.</td>
</tr>
<tr>
<td>ToString(string) method</td>
<td>The equivalent to DateTime.ToString(format) with the time zone offset appended to the end.</td>
</tr>
<tr>
<td>ToString(string, IFormatProvider) method</td>
<td>The equivalent to DateTime.ToString(format, provider) with the time zone offset appended to the end.</td>
</tr>
</tbody>
</table>
**ToString() method**

The equivalent to DateTime.ToString() with the time zone offset appended to the end.

**Visual Basic syntax**

```vbnet
Public Overrides Function ToString() As String
```

**C# syntax**

```csharp
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**

```vbnet
Public Function ToString(provider As IFormatProvider) As String
```

**C# syntax**

```csharp
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**

```vbnet
Public Function ToString(format As String) As String
```

**C# syntax**

```csharp
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**

```vbnet
Public Function ToString(format As String, provider As IFormatProvider) As String
```

**C# syntax**

```csharp
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.Month.

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 the DBCommand interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close method</td>
<td>Closes the current SQL statement or command.</td>
</tr>
<tr>
<td>ExecuteNonQuery method</td>
<td>Executes a non-query statement.</td>
</tr>
<tr>
<td>ExecuteReader method</td>
<td>Executes a query statement returning the result set.</td>
</tr>
<tr>
<td>Prepare method</td>
<td>Prepares the SQL statement stored in CommandText for execution.</td>
</tr>
<tr>
<td>CommandText property</td>
<td>The SQL statement to be executed.</td>
</tr>
<tr>
<td>Parameters property</td>
<td>Gets the DBParameterCollection for this DBCommand.</td>
</tr>
</tbody>
</table>

Remarks

DBCommand can represent an update or query.

Example

The following C# code uses the DBCommand interface to execute two queries:

```csharp
DBCommand stmt = conn.CreateCommand();
stmt.CommandText = "SELECT t1a1, t1a2 FROM table1 ";
DBRowReader rs = stmt.ExecuteReader();
printResultSet( rs );
```
rs.Close();
stmt.CommandText = "SELECT t2a1 FROM table2 ";
rs = stmt.ExecuteReader();
printResultSet(rs);
rs.Close();
stmt.Close();

The following C# code uses the DBCommand interface to execute an update with parameters:

```csharp
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 = SqlDbType.SQL_CHAR;
    param.Value = "10000";
    cstmt.Parameters.Add(param);
    param = new DBParameter();
    param.DbType = SqlDbType.SQL_INTEGER;
    param.Value = 20000;
    cstmt.Parameters.Add(param);
    param = new DBParameter();
    param.DbType = SqlDbType.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 [MobiLink server .NET]” on page 534

DBConnection interface

Represents a MobiLink ODBC connection.

Visual Basic syntax

Public Interface DBConnection

C# syntax

public interface DBConnection

Members

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close method</td>
<td>Closes the current connection.</td>
</tr>
<tr>
<td>Commit method</td>
<td>Commits the current transaction.</td>
</tr>
<tr>
<td>CreateCommand method</td>
<td>Creates a SQL statement or command on this connection.</td>
</tr>
<tr>
<td>Rollback method</td>
<td>Rolls back the current transaction.</td>
</tr>
</tbody>
</table>

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

```csharp
public interface DBConnectionContext
```

Members

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetConnection method</td>
<td>Returns the existing connection to the MobiLink consolidated database.</td>
</tr>
<tr>
<td>GetDownloadData method</td>
<td>Returns the DownloadData for the current synchronization.</td>
</tr>
<tr>
<td>GetProperties method</td>
<td>Returns a collection of properties based on this connection script version.</td>
</tr>
<tr>
<td>GetRemoteID method</td>
<td>Returns the remote ID of the database currently synchronizing on this connection.</td>
</tr>
<tr>
<td>GetServerContext method</td>
<td>Returns the current server context.</td>
</tr>
<tr>
<td>GetVersion method</td>
<td>Returns the version string for this connection.</td>
</tr>
<tr>
<td>NetworkData property</td>
<td>Returns information about the network streams for a synchronization.</td>
</tr>
</tbody>
</table>

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 502.

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.

```csharp
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

● “ServerContext.MakeConnection method [MobiLink server .NET]” on page 564

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`:

```csharp
// 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**

```vbnet
Public Function GetProperties() As NameValueCollection
```

**C# syntax**

```csharp
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 621.

**Example**

The following example shows how to output the properties for a `DBConnectionContext`. This example assumes you have a `DBConnectionContext` instance named `_cc`.

```csharp
// 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**

```vbnet
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:

```csharp
// 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:

```csharp
// 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, see “ml_add_property system procedure” on page 621.

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;
        }
    }

NetworkData property
Returns information about the network streams for a synchronization.

Visual Basic syntax
Public ReadOnly Property NetworkData As NetworkData

C# syntax
public NetworkData NetworkData {get;}

Returns
Information about the network streams used for the request, or null if the collection has not been enabled.
Remarks

This method is useful when authenticating against another server in the enterprise that uses the client-side certificate and HTTP headers.

To enable a collection of network stream data, add collect_network_data=1 to your -x switches. This option adds additional per-sync memory overhead to store the data.

See also

● “NetworkData interface [MobiLink server .NET]” on page 555

**DBParameter class**

Represents a bound ODBC parameter.

**Visual Basic syntax**

Public Class DBParameter

**C# syntax**

public class DBParameter

**Members**

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DbType property</td>
<td>The SQLType of this parameter.</td>
</tr>
<tr>
<td>Direction property</td>
<td>The Input/Output direction of this parameter.</td>
</tr>
<tr>
<td>IsNullable property</td>
<td>True if the parameter can be null; otherwise, false.</td>
</tr>
<tr>
<td>ParameterName property</td>
<td>The name of this parameter.</td>
</tr>
<tr>
<td>Precision property</td>
<td>The Decimal precision of this parameter.</td>
</tr>
<tr>
<td>Scale property</td>
<td>The resolvable digits of this parameter.</td>
</tr>
<tr>
<td>Size property</td>
<td>The size of this parameter, measured in bytes.</td>
</tr>
<tr>
<td>Value property</td>
<td>The value of this parameter.</td>
</tr>
<tr>
<td>HasChanged field</td>
<td>Returns whether the parameter has been modified since creation.</td>
</tr>
</tbody>
</table>

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:

```csharp
using( DBCommand cstmt = conn.CreateCommand() ) {
    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();
}
```

**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 SqlDbType.SQL_NUMERIC and SqlDbType.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 the DBParameterCollection class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBParameterCollection constructor</td>
<td>Creates an empty list of DBParameters.</td>
</tr>
<tr>
<td>Add method</td>
<td>Adds the given parameter to the collection.</td>
</tr>
<tr>
<td>Clear method</td>
<td>Removes all parameters from the collection.</td>
</tr>
<tr>
<td>Contains method</td>
<td>Checks whether the collection contains a parameter with the specified name.</td>
</tr>
<tr>
<td>CopyTo method</td>
<td>Copies the contents of the collection into the given array starting at the specified index.</td>
</tr>
</tbody>
</table>
### 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.

### 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 at the given index in the collection.

### Remarks
A DBParameterCollection 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
```vbnet
Public Sub New()
```

#### C# syntax
```csharp
public DBParameterCollection()
```

### Add method
Adds the given parameter to the collection.

#### Visual Basic syntax
```vbnet
Public Function Add(ByVal value As Object) As Integer
```
C# syntax

```csharp
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 [MobiLink server .NET]” on page 530

**Clear method**
Removes all parameters from the collection.

**Visual Basic syntax**

```vbnet
Public Sub Clear()
```

**C# syntax**

```csharp
public void Clear()
```

**Contains method**
Checks whether the collection contains a parameter with the specified name.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contains(object) method</td>
<td>Returns true if the collection contains the given DBParameter.</td>
</tr>
<tr>
<td>Contains(string) method</td>
<td>Checks whether the collection contains a parameter with the specified name.</td>
</tr>
</tbody>
</table>

**Contains(object) method**
Returns true if the collection contains the given DBParameter.

**Visual Basic syntax**

```vbnet
Public Function Contains(ByVal value As Object) As Boolean
```

**C# syntax**

```csharp
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 [MobiLink server .NET]” on page 530

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>IndexOf(object) method</td>
<td>Returns the index of the given DBParameter in the collection.</td>
</tr>
<tr>
<td>IndexOf(string) method</td>
<td>Returns the index of the parameter with the given name in the collection.</td>
</tr>
</tbody>
</table>

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 [MobiLink server .NET]” on page 530

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 [MobiLink server .NET]” on page 530

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 [MobiLink server .NET]” on page 530

**RemoveAt method**

Removes the parameter with the given name from the collection.

**Overload list**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RemoveAt(int) method</td>
<td>Removes the DBParameter at the given index in the collection.</td>
</tr>
<tr>
<td>RemoveAt(string) method</td>
<td>Removes the parameter with the given name from the collection.</td>
</tr>
</tbody>
</table>

**RemoveAt(int) method**

Removes the DBParameter at the given index in the collection.

**Visual Basic syntax**

```vbnet
Public Sub RemoveAt(ByVal index As Integer)
```

**C# syntax**

```csharp
public void RemoveAt(int index)
```

**Parameters**

- index The index of the DBParameter to remove.

**See also**

- “DBParameter class [MobiLink server .NET]” on page 530

**RemoveAt(string) method**

Removes the parameter with the given name from the collection.

**Visual Basic syntax**

```vbnet
Public Sub RemoveAt(ByVal parameterName As String)
```

**C# syntax**

```csharp
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**

```vbnet
Public ReadOnly Property Count As Integer
```

**C# syntax**

```csharp
public int Count {get;}
```

**IsFixedSize property**

Returns false.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property IsFixedSize As Boolean
```

**C# syntax**

```csharp
public bool IsFixedSize {get;}
```

**IsReadOnly property**

Returns false.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property IsReadOnly As Boolean
```

**C# syntax**

```csharp
public bool IsReadOnly {get;}
```

**IsSynchronized property**

Returns false.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property IsSynchronized As Boolean
```

**C# syntax**

```csharp
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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>this[int] property</td>
<td>Gets or sets the DBParameter at the given index in the collection.</td>
</tr>
<tr>
<td>this[string] property</td>
<td>Gets or sets the DBParameter with the given name in the collection.</td>
</tr>
</tbody>
</table>

**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 [MobiLink server .NET]” on page 530

**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

```csharp
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 [MobiLink server .NET]” on page 530

**DBRowReader interface**

Represents a set of rows being read from a database.

Visual Basic syntax

```vbnet
Public Interface DBRowReader
```

C# syntax

```csharp
public interface DBRowReader
```

Members

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close method</td>
<td>Cleans up all resources used by this MLDBRowReader.</td>
</tr>
<tr>
<td>NextRow method</td>
<td>Retrieves and returns the next row in the result set.</td>
</tr>
<tr>
<td>ColumnNames property</td>
<td>Gets the names of all columns in the result set.</td>
</tr>
<tr>
<td>ColumnTypes property</td>
<td>Gets the types of all columns in the result set.</td>
</tr>
</tbody>
</table>

Remarks

Executing the ExecuteReader method creates a DBRowReader.

See also

- “DBCommand.ExecuteReader method [MobiLink server .NET]” on page 522

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 [MobiLink server .NET]” on page 581

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 the DownloadData interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetDownloadTableByName method</td>
<td>Gets the named download table for this synchronization.</td>
</tr>
<tr>
<td>GetDownloadTables method</td>
<td>Gets an array of all the tables for download in this synchroniza-</td>
</tr>
<tr>
<td></td>
<td>tion.</td>
</tr>
</tbody>
</table>

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 341 and “Direct row handling” on page 585.

See also

- “DownloadTableData interface [MobiLink server .NET]” on page 547
- “DBConnectionContext.GetDownloadData method [MobiLink server .NET]” on page 526

**GetDownloadTableByName method**

Gets the named download table for this synchronization.

**Visual Basic syntax**

```vbnet
Public Function GetDownloadTableByName(
    ByVal table_name As String
) As DownloadTableData
```

**C# syntax**

```csharp
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**

```vbnet
Public Function GetDownloadTables() As DownloadTableData()
```

**C# syntax**

```csharp
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 DBCConnectionContext instance named _cc.

```csharp
// 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**

```vbnet
Public Interface DownloadTableData
```

**C# syntax**

```csharp
public interface DownloadTableData
```

**Members**

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetDeleteCommand method</td>
<td>Get a command which allows the user to add delete operations to the download data operations.</td>
</tr>
<tr>
<td>GetLastDownloadTime method</td>
<td>Returns the last download time for this table.</td>
</tr>
<tr>
<td>GetName method</td>
<td>Gets the table name of this instance.</td>
</tr>
<tr>
<td>GetSchemaTable method</td>
<td>Gets a DataTable that describes the metadata for this download table.</td>
</tr>
<tr>
<td>GetUpsertCommand method</td>
<td>Get a command which allows the user to add upsert(insert/update) operations to the download data operations.</td>
</tr>
</tbody>
</table>

**Remarks**

Use this interface to set the data operations that are downloaded to a synchronization client site.
Example

Suppose you have the following table:

```sql
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:

```java
// The method used for the handle_DownloadData event
public void HandleDownload() {
    // _cc is a DBConnectionContext instance.
    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.

```java
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.

```java
void SetDownloadDeletes(DownloadTableData td) {
    IDbCommand delete_cmd = td.GetDeleteCommand();

    // The following method calls are the same as the following SQL statement:
```
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 [MobiLink server .NET]” on page 547

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**

```vbnet
Public Function GetName() As String
```

**C# syntax**

```csharp
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**

```vbnet
Public Function GetSchemaTable() As DataTable
```

**C# syntax**

```csharp
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**

```vbnet
Public Function GetUpsertCommand() As IDbCommand
```

**C# syntax**

```csharp
public IDbCommand GetUpsertCommand()
```
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 [MobiLink server .NET]” on page 547

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 the FatalException class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FatalException constructor</td>
<td>Creates a new FatalException with the default message.</td>
</tr>
</tbody>
</table>

FatalException constructor
    Creates a new FatalException with the default message.
## Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FatalException() constructor</td>
<td>Creates a new FatalException with the default message.</td>
</tr>
<tr>
<td>FatalException(string) constructor</td>
<td>Creates a new FatalException with the given message.</td>
</tr>
<tr>
<td>FatalException(string, Exception) constructor</td>
<td>Creates a new FatalException with the given message and containing the given inner exception that caused this one.</td>
</tr>
</tbody>
</table>

### FatalException() constructor

Creates a new FatalException with the default message.

**Visual Basic syntax**

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

**C# syntax**

```csharp
public FatalException()
```

### FatalException(string) constructor

Creates a new FatalException with the given message.

**Visual Basic syntax**

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

**C# syntax**

```csharp
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**

```vbnet
Public Sub New(ByVal message As String, ByVal ie As Exception)
```

**C# syntax**

```csharp
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 the LogMessage class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LogMessage constructor</strong></td>
<td>Create a LogMessage with the given attributes.</td>
</tr>
<tr>
<td><strong>MessageType enumeration</strong></td>
<td>Enumerates all possible LogMessage types.</td>
</tr>
<tr>
<td><strong>Text property</strong></td>
<td>The main text of the message.</td>
</tr>
<tr>
<td><strong>Type property</strong></td>
<td>The type of log message that this instance represents.</td>
</tr>
<tr>
<td><strong>User property</strong></td>
<td>The user for which this message is being logged.</td>
</tr>
</tbody>
</table>

**Remarks**
  An instance of this class is passed into a LogCallback.

**See also**
  ● “LogCallback delegate [MobiLink server .NET]” on page 581

**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**

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ERROR</td>
<td>A log message is an error.</td>
</tr>
<tr>
<td>WARNING</td>
<td>A log message is a warning.</td>
</tr>
<tr>
<td>INFO</td>
<td>A log information message.</td>
</tr>
</tbody>
</table>

**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;}

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
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.

NetworkData interface

Contains information about the network streams for a synchronization.

Visual Basic syntax

Public Interface NetworkData

C# syntax

public interface NetworkData

Members

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetHTTPHeaderValue method</td>
<td>Returns the value of the last header received by the server with the supplied name.</td>
</tr>
<tr>
<td>GetHTTPHeaderValues method</td>
<td>Returns all the header values received by the server associated with the supplied name.</td>
</tr>
<tr>
<td>ClientCertificates property</td>
<td>Returns an X509Certificate2Collection containing any certificates sent by the client.</td>
</tr>
<tr>
<td>HTTPHeaders property</td>
<td>Returns a dictionary that maps header names to a list of header values.</td>
</tr>
<tr>
<td>IsEndToEndEncrypted property</td>
<td>Determines if this synchronization is end-to-end encrypted.</td>
</tr>
<tr>
<td>IsHTTP property</td>
<td>Determines if the synchronization uses HTTP or HTTPS.</td>
</tr>
<tr>
<td>IsTLS property</td>
<td>Determines if this synchronization uses TLS.</td>
</tr>
</tbody>
</table>
Remarks

This interface is useful when authenticating against another server in the enterprise that uses the client-side certificate and HTTP headers.

To enable a collection of network stream data, add collect_network_data=1 to your -x switches. This option adds additional per-sync memory overhead to store the data. When using TLS or HTTPS with client-side certificates, add trusted_certificates= certificate file; to have the server ask the client to send a certificate during the TLS handshake, incurring a time and network cost.

You can obtain a NetworkData object by invoking the NetworkData method of the DBConnectionContext interface. When using HTTP or HTTPS, it contains the header data for the last HTTP request received by the server before the authenticate scripts are invoked.

Example

The following example illustrates how to get a NetworkData object from the DBConnectionContext object, and output the data.

```csharp
using iAnywhere.MobiLink.Script;
using System.Collections.Generic;

public class OrderProcessor {
    DBConnectionContext _cc;

    public OrderProcessor( DBConnectionContext cc ) {
        _cc = cc;
    }

    public void AuthUser() {
        NetworkData nd = _cc.NetworkData;
        if( nd != null ) {
            if( nd.IsHTTP ) {
                string user_agent = nd.GetHTTPHeaderValue( "user-agent" );
                PrintLn( "  user-agent: " + user_agent.Substring( 0, user_agent.IndexOf( '/' ) ) );
            } else {
                PrintLn( "no http" );
            }
            if( nd.IsTLS ) {
                PrintLn( "tls" );
                X509Certificate2Collection certs = nd.ClientCertificates;
                if( certs != null ) {
                    int n = 1;
                    foreach( X509Certificate2 x509 in certs ) {
                        PrintLn( "    cert " + n++ );
                        PrintLn( "      Subject: " + x509.SubjectName.Name );
                        PrintLn( "      Issuer: " + x509.IssuerName.Name );
                    }
                } else {
                    PrintLn( "  no client cert" );
                }
            } else {
                PrintLn( "no tls" );
            }
            if( nd.IsEndToEndEncrypted ) {
                PrintLn( "e2ee" );
            }
        }
    }
}
```
GetHTTPHeaderValue method

Returns the value of the last header received by the server with the supplied name.

Visual Basic syntax

Public Function GetHTTPHeaderValue(ByVal name As String) As String

C# syntax

public string GetHTTPHeaderValue(string name)

Parameters

- **name**  The header name to return the value for.

Returns

The last header value associated with the supplied header name.

See also

- “NetworkData.GetHTTPHeaderValues method [MobiLink server .NET]” on page 557
- “NetworkData.HTTPHeaders property [MobiLink server .NET]” on page 558

GetHTTPHeaderValues method

Returns all the header values received by the server associated with the supplied name.

Visual Basic syntax

Public Function GetHTTPHeaderValues(ByVal name As String) As IList< string >

C# syntax

public IList< string > GetHTTPHeaderValues(string name)

Parameters

- **name**  The header name to return the values for.

Returns

The header values associated with the supplied header name.
See also
  ● “NetworkData.GetHTTPHeaderValue method [MobiLink server .NET]” on page 557
  ● “NetworkData.HTTPHeaders property [MobiLink server .NET]” on page 558

**ClientCertificates property**

Returns an X509Certificate2Collection containing any certificates sent by the client.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property ClientCertificates As X509Certificate2Collection
```

**C# syntax**

```csharp
public X509Certificate2Collection ClientCertificates {get;}
```

**Returns**

An X509Certificate2Collection containing the X509 certificates that identify the client, or null if no such certificates were provided.

**Remarks**

This function will return a non-null value only if isTLS() is true, and the client supplies a certificate using the "identity" stream parameter, and the trusted_certificates option is set on the server. A non-null CertPath will contain the certificates in order, from the self-signed certificate to the peer certificate.

See also
  ● “trusted_certificates example for .NET” on page 93

**HTTPHeaders property**

Returns a dictionary that maps header names to a list of header values.

**Visual Basic syntax**

```vbnet
Public ReadOnly Property HTTPHeaders As IDictionary(Of String, IList(Of String))
```

**C# syntax**

```csharp
public IDictionary<string, IList<string>> HTTPHeaders {get;}
```

**Returns**

A dictionary of header name-value pairs.

See also
  ● “NetworkData.GetHTTPHeaderValue method [MobiLink server .NET]” on page 557
  ● “NetworkData.GetHTTPHeaderValues method [MobiLink server .NET]” on page 557
IsEndToEndEncrypted property
Determine if this synchronization is end-to-end encrypted.

Visual Basic syntax
Public ReadOnly Property IsEndToEndEncrypted As Boolean

C# syntax
public bool IsEndToEndEncrypted {get;}

IsHTTP property
Determines if the synchronization uses HTTP or HTTPS.

Visual Basic syntax
Public ReadOnly Property IsHTTP As Boolean

C# syntax
public bool IsHTTP {get;}

Returns
True if this synchronization uses HTTP or HTTPS; otherwise, returns false.

IsTLS property
Determines if this synchronization uses TLS.

Visual Basic syntax
Public ReadOnly Property IsTLS As Boolean

C# syntax
public bool IsTLS {get;}

Returns
True if this synchronization uses TLS; otherwise, returns false.

ScriptExecutionException class
Signals that an error has occurred in a user script.

Visual Basic syntax
Public Class ScriptExecutionException
    Inherits System.ApplicationException

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0 559
C# syntax

```csharp
public class ScriptExecutionException : System.ApplicationException
```

Base classes

- `System.ApplicationException`

Derived classes

- “ServerException class [MobiLink server .NET]” on page 566
- “SynchronizationException class [MobiLink server .NET]” on page 570

Members

All members of the `ScriptExecutionException` class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ScriptExecutionException constructor</code></td>
<td>Creates a new <code>ScriptExecutionException</code> with the default message.</td>
</tr>
</tbody>
</table>

Remarks

Throwing this exception or any derivations of it, except for `SynchronizationException`, causes the MobiLink server to shut down.

**ScriptExecutionException constructor**

Creates a new `ScriptExecutionException` with the default message.

Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>ScriptExecutionException() constructor</code></td>
<td>Creates a new <code>ScriptExecutionException</code> with the default message.</td>
</tr>
<tr>
<td><code>ScriptExecutionException(string) constructor</code></td>
<td>Creates a new <code>ScriptExecutionException</code> with the given message.</td>
</tr>
<tr>
<td><code>ScriptExecutionException(string, Exception) constructor</code></td>
<td>Creates a new <code>ScriptExecutionException</code> with the given message and containing the given inner exception that caused this one.</td>
</tr>
</tbody>
</table>

**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 the ServerContext interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>getProperties method</td>
<td>Returns a collection of the properties for the given component and set.</td>
</tr>
<tr>
<td>getPropertiesByVersion method</td>
<td>Returns a collection of the properties for the given script version.</td>
</tr>
<tr>
<td>getPropertySetNames method</td>
<td>Returns a collection of the set names for the given component.</td>
</tr>
<tr>
<td>GetStartClassInstances method</td>
<td>Returns all start classes loaded into this domain.</td>
</tr>
<tr>
<td>MakeConnection method</td>
<td>Creates a new database connection.</td>
</tr>
<tr>
<td>Shutdown method</td>
<td>Causes the MobiLink server to perform a soft shutdown.</td>
</tr>
<tr>
<td>ErrorListener event</td>
<td>Triggered when the MobiLink server prints an error.</td>
</tr>
<tr>
<td>InfoListener event</td>
<td>Triggered when the MobiLink server prints information.</td>
</tr>
<tr>
<td>ShutdownListener event</td>
<td>Triggered when the MobiLink server is shutting down.</td>
</tr>
<tr>
<td>WarningListener event</td>
<td>Triggered when the MobiLink server prints a warning.</td>
</tr>
</tbody>
</table>

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
- “DBConnectionContext.GetServerContext method [MobiLink server .NET]” on page 528

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 column of the ml_property table.
- **set**  Refers to the property_set_name column 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, see “ml_add_property system procedure” on page 621.

**getPropertiesByVersion method**

Returns a collection of the properties for the given script version.

**Visual Basic syntax**

```vbnet
Public Function getPropertiesByVersion(
    ByVal script_version As String)
    As NameValueCollection
End Function
```

**C# syntax**

```csharp
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, see “ml_add_property system procedure” on page 621.

**getPropertySetName method**

Returns a collection of the set names for the given component.

**Visual Basic syntax**

```vbnet
Public Function getPropertySetName(
    ByVal component As String)
    As StringCollection
End Function
```

**C# syntax**

```csharp
public StringCollection getPropertySetName(string component)
```
Parameters

- **component**  Refers to the component_name column 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, see “ml_add_property system procedure” on page 621.

GetStartClassInstances method

Returns all start classes loaded into this domain.

Visual Basic syntax

```vbnet
Public Function GetStartClassInstances() As Object()
```

C# syntax

```csharp
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 503.

Example

The following example demonstrates how to find a start class:

```csharp
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

```vbnet
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 a 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 [MobiLink server .NET]” on page 559

Members
All members of the ServerException class, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScriptExecutionException con-structor</td>
<td>Creates a new ScriptExecutionException with the default message.</td>
</tr>
<tr>
<td>ServerException constructor</td>
<td>Creates a new ServerException with the default message.</td>
</tr>
</tbody>
</table>

ServerException constructor
Creates a new ServerException with the default message.
Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ServerException() constructor</td>
<td>Creates a new ServerException with the default message.</td>
</tr>
<tr>
<td>ServerException(string) constr-</td>
<td>Creates a new ServerException with the given message.</td>
</tr>
<tr>
<td>uctor</td>
<td></td>
</tr>
<tr>
<td>ServerException(string, Exception) constructor</td>
<td>Creates a new ServerException with the given message and containing the given inner exception that caused this one.</td>
</tr>
</tbody>
</table>

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**

```vbnet
Public NotInheritable Class SpatialUtilities

C# syntax

```net
public sealed class SpatialUtilities
```

**Members**

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateSpatialValue method</td>
<td>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.</td>
</tr>
<tr>
<td>GetBytes method</td>
<td>Returns a new byte array with the same spatial data as the given byte array, but with the SRID removed.</td>
</tr>
<tr>
<td>GetSRID method</td>
<td>Returns the SRID for the given spatial value.</td>
</tr>
<tr>
<td>SetSRID method</td>
<td>Stores the given SRID in the first four bytes of the given byte array.</td>
</tr>
</tbody>
</table>

**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**

```vbnet
Public Shared Function CreateSpatialValue(
    ByVal srid As Integer,
    ByVal spatial_value As Byte())
    As Byte()
```

**C# syntax**

```csharp
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 [MobiLink server .NET]” on page 559

Members

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScriptExecutionException constructor</td>
<td>Creates a new ScriptExecutionException with the default message.</td>
</tr>
<tr>
<td>SynchronizationException constructor</td>
<td>Creates a new SynchronizationException with the default message.</td>
</tr>
</tbody>
</table>

SynchronizationException constructor

Creates a new SynchronizationException with the default message.
Overload list

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SynchronizationException() constructor</td>
<td>Creates a new SynchronizationException with the default message.</td>
</tr>
<tr>
<td>SynchronizationException(string) constructor</td>
<td>Creates a new SynchronizationException with the given message.</td>
</tr>
<tr>
<td>SynchronizationException(string, Exception) constructor</td>
<td>Creates a new SynchronizationException with the given message and containing the given inner exception that caused this one.</td>
</tr>
</tbody>
</table>

**SynchronizationException() constructor**

Creates a new SynchronizationException with the default message.

**Visual Basic syntax**

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

**C# syntax**

```csharp
public SynchronizationException()
```

**SynchronizationException(string) constructor**

Creates a new SynchronizationException with the given message.

**Visual Basic syntax**

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

**C# syntax**

```csharp
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**

```vbnet
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 the UpdateDataReader interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetSchemaTable method (Inherited from System.Data.IDataReader)</td>
<td>Returns a System.Data.DataTable that describes the column metadata of the System.Data.IDataReader.</td>
</tr>
<tr>
<td>NextResult method (Inherited from System.Data.IDataReader)</td>
<td>Advances the data reader to the next result, when reading the results of batch SQL statements.</td>
</tr>
<tr>
<td>Read method (Inherited from System.Data.IDataReader)</td>
<td>Advances the System.Data.IDataReader to the next record.</td>
</tr>
<tr>
<td>SetNewRowValues method</td>
<td>Sets the mode of this DataReader to return new column values (the post-updated row).</td>
</tr>
<tr>
<td>SetOldRowValues method</td>
<td>Sets the mode of this DataReader to return old column values (the pre-updated row).</td>
</tr>
<tr>
<td>Depth property (Inherited from System.Data.IDataReader)</td>
<td>Gets a value indicating the depth of nesting for the current row.</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>IsClosed property (Inherited from System.Data.IDataReader)</td>
<td>Gets a value indicating whether the data reader is closed.</td>
</tr>
<tr>
<td>RecordsAffected property</td>
<td>Gets the number of rows changed, inserted, or deleted by execution of the SQL statement.</td>
</tr>
</tbody>
</table>

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**

```vbnet
Public Sub SetNewRowValues()
```

**C# syntax**

```csharp
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**

```vbnet
Public Sub SetOldRowValues()
```

**C# syntax**

```csharp
public void SetOldRowValues()
```

**UploadData interface**
Encapsulates upload operations for direct row handling.

**Visual Basic syntax**

```vbnet
Public Interface UploadData
```

**C# syntax**

```csharp
public interface UploadData
```
Members

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetUploadedTableByName method</td>
<td>Gets the named Uploaded table data in this uploaded transaction.</td>
</tr>
<tr>
<td>GetUploadedTables method</td>
<td>Gets an array of all the uploaded tables data in this uploaded transaction.</td>
</tr>
</tbody>
</table>

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 352.

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 we 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:

```csharp
// 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.

See also

- “UploadedTableData interface [MobiLink server .NET]” on page 575

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:

```csharp
// 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 the UploadedTableData interface, including all inherited members.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GetDeletes method</td>
<td>Gets a DataReader with the deletes for this uploaded table data.</td>
</tr>
<tr>
<td>GetInserts method</td>
<td>Gets a DataReader with the inserts for this uploaded table data.</td>
</tr>
<tr>
<td>GetName method</td>
<td>Gets the table name of this instance.</td>
</tr>
<tr>
<td>GetSchemaTable method</td>
<td>Gets a DataTable that describes the metadata for this download table.</td>
</tr>
<tr>
<td>GetUpdates method</td>
<td>Gets a DataReader with the updates for this uploaded table data.</td>
</tr>
</tbody>
</table>

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.

```csharp
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(data_reader.GetString(1)); // pcol3
            row_str.Append(data_reader.GetString(2)); // col2
            writer.WriteLine(row_str);
        }
    }
}
```

GetInserts method

Gets a DataReader with the inserts for this uploaded table data.

Visual Basic syntax

```vbnet
Public Function GetInserts() As IDataReader
```

C# syntax

```csharp
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.

**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:

```csharp
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()) {
            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("<NULL>");
            } 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("<NULL>");
            } 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
● “ServerContext.ErrorListener event [MobiLink server .NET]” on page 565
● “ServerContext.InfoListener event [MobiLink server .NET]” on page 565
● “ServerContext.WarningListener event [MobiLink server .NET]” on page 566

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
● “ServerContext.ShutdownListener event [MobiLink server .NET]” on page 565

SQLType enumeration
Enumerates all possible ODBC data types.

Visual Basic syntax
Public Enum SQLType
C# syntax

```csharp
public enum SQLType
```

## Members

<table>
<thead>
<tr>
<th>Member name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL_TYPE_NULL</td>
<td>Null data type.</td>
</tr>
<tr>
<td>SQL_UNKNOWN_TYPE</td>
<td>Unknown data type.</td>
</tr>
<tr>
<td>SQL_CHAR</td>
<td>UTF-8 character array of a set size. Has .NET type String.</td>
</tr>
<tr>
<td>SQL_NUMERIC</td>
<td>Numeric value of set size and precision. Has .NET type Decimal.</td>
</tr>
<tr>
<td>SQL_DECIMAL</td>
<td>Decimal number of set size and precision. Has .NET type Decimal.</td>
</tr>
<tr>
<td>SQL_INTEGER</td>
<td>32-bit integer. Has .NET type Int32.</td>
</tr>
<tr>
<td>SQL_SMALLINT</td>
<td>16-bit integer. Has .NET type Int16</td>
</tr>
<tr>
<td>SQL_FLOAT</td>
<td>Floating point number with ODBC driver defined precision. Has .NET type Double.</td>
</tr>
<tr>
<td>SQL_REAL</td>
<td>Single precision floating point number. Has .NET type Single.</td>
</tr>
<tr>
<td>SQL_DOUBLE</td>
<td>Double precision floating point number. Has .NET type Double.</td>
</tr>
<tr>
<td>SQL_DATE</td>
<td>A date. Has .NET type DateTime.</td>
</tr>
<tr>
<td>SQL_DATETIME</td>
<td>A date and time. Has .NET type DateTime.</td>
</tr>
<tr>
<td>Member name</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SQL_TIME</td>
<td>A time.</td>
</tr>
<tr>
<td></td>
<td>Has .NET type DateTime</td>
</tr>
<tr>
<td>SQL_INTERVAL</td>
<td>An interval of time.</td>
</tr>
<tr>
<td></td>
<td>Has .NET type TimeSpan</td>
</tr>
<tr>
<td>SQL_TIMESTAMP</td>
<td>A time stamp.</td>
</tr>
<tr>
<td></td>
<td>Has .NET type DateTime</td>
</tr>
<tr>
<td>SQL_VARCHAR</td>
<td>A null terminated UTF-8 string with a user set maximum length.</td>
</tr>
<tr>
<td></td>
<td>Has .NET type String</td>
</tr>
<tr>
<td>SQL_TYPE_DATE</td>
<td>A date.</td>
</tr>
<tr>
<td></td>
<td>Has .NET type DateTime</td>
</tr>
<tr>
<td>SQL_TYPE_TIME</td>
<td>A time.</td>
</tr>
<tr>
<td></td>
<td>Has .NET type DateTime</td>
</tr>
<tr>
<td>SQL_TYPE_TIMESTAMP</td>
<td>A timestamp.</td>
</tr>
<tr>
<td></td>
<td>Has .NET type DateTime</td>
</tr>
<tr>
<td>SQL_DEFAULT</td>
<td>A default type.</td>
</tr>
<tr>
<td></td>
<td>Has no type.</td>
</tr>
<tr>
<td>SQL_ARD_TYPE</td>
<td>An ARD object.</td>
</tr>
<tr>
<td></td>
<td>Has no type.</td>
</tr>
<tr>
<td>SQL_BIT</td>
<td>A single bit.</td>
</tr>
<tr>
<td></td>
<td>Has .NET type Boolean</td>
</tr>
<tr>
<td>SQL_TINYINT</td>
<td>8-bit integer.</td>
</tr>
<tr>
<td></td>
<td>Has .NET type SByte</td>
</tr>
<tr>
<td>SQL_BIGINT</td>
<td>64-bit integer.</td>
</tr>
<tr>
<td></td>
<td>Has .NET type Int64</td>
</tr>
<tr>
<td>Member name</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>SQL_LONGVARBINARY</td>
<td>Variable length binary data with a driver dependent maximum length. Has .NET type byte[].</td>
</tr>
<tr>
<td>SQL_VARBINARY</td>
<td>Variable length binary data with a user specified maximum length. Has .NET type byte[].</td>
</tr>
<tr>
<td>SQL_BINARY</td>
<td>Fixed length binary data. Has .NET type byte[].</td>
</tr>
<tr>
<td>SQL_LONGVARCHAR</td>
<td>A null terminated UTF-8 string with a driver dependent maximum length. Has .NET type String.</td>
</tr>
<tr>
<td>SQL_GUID</td>
<td>A GUID. Has .NET type Guid.</td>
</tr>
<tr>
<td>SQL_WCHAR</td>
<td>Unicode character array of fixed size. Has .NET type String.</td>
</tr>
<tr>
<td>SQL_WVARCHAR</td>
<td>Null terminated Unicode string of user defined maximum length; Has .NET type String.</td>
</tr>
<tr>
<td>SQL_WLONGVARCHAR</td>
<td>Null terminated Unicode string of driver dependent maximum length; Has .NET type String.</td>
</tr>
<tr>
<td>SQL_SS_TIMESTAMPPOFFSET</td>
<td>Timestamp with time zone offset; Has .NET type iAnywhere.MobiLink.Script.DateTimeWithTimeZone This can be used only against Microsoft SQL Server and Oracle databases.</td>
</tr>
</tbody>
</table>

**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:
  - “Direct uploads” on page 588
  - “handle_UploadData connection event” on page 352

- **handle_DownloadData** allows you to set download operations using the DownloadData interface. See:
  - “Direct downloads” on page 594
  - “handle_DownloadData connection event” on page 341

Components of the MobiLink server API for direct row handling

For the Java API:

- “DBConnectionContext.getDownloadData method [MobiLink server Java]”
- “DownloadData interface [MobiLink server Java]”
- “DownloadTableData interface [MobiLink server Java]”
- “UpdateResultSet interface [MobiLink server Java]”
- “UploadData interface [MobiLink server Java]”
- “UploadedTableData interface [MobiLink server Java]”

For the .NET API:

- “DBConnectionContext.GetDownloadData method [MobiLink server .NET]”
- “DownloadData interface [MobiLink server .NET]”
- “DownloadTableData interface [MobiLink server .NET]”
- “UpdateDataReader interface [MobiLink server .NET]”
- “UploadedTableData interface [MobiLink server .NET]”
- “UploadData interface [MobiLink server .NET]”

Direct row handling setup

To use direct row handling, you need familiarity with how to create a MobiLink consolidated database, add synchronization scripts, and create MobiLink remote users.

Following is an overview of how to synchronize with a data source other than a consolidated database.
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. To handle uploads, write a public method using the UploadData interface and register it for the handle_UploadData connection event.

   See “Direct uploads” on page 588.

3. To handle downloads, write a public method using the DownloadData interface and register it for the handle_DownloadData connection event (or another event).

   See “Direct downloads” on page 594.

Other resources for getting started

- “Tutorial: Using direct row handling” [MobiLink - Getting Started]
- http://www.sybase.com/detail?id=1058600#319 (You need a Sybase.com login to view this page.)
- “Setting up Java synchronization logic”
- “Implementing synchronization scripts in .NET”


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, one column must contain unique, unchanging values that identify the row.

See “Unique primary keys” on page 108.

Column Names

The column names of tables are always sent from the client and can be used for direct row handling. Alternatively, you can use column indexes to access row information, based on the column order sent up from the remote database.

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 “Implementing timestamp-based downloads” on page 99.
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.

Direct uploads

To handle direct uploads:

1. Register a Java or .NET method for the handle_UploadData connection event. See “handle_UploadData connection event” on page 352.

2. Write a method for the handle_UploadData synchronization event. This event accepts one UploadData parameter. See:
   - “UploadData interface [MobiLink server Java]” on page 489
   - “UploadData interface [MobiLink server .NET]” on page 573

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:

- “Synchronization script writing in Java” on page 427
- “Synchronization scripts in .NET” on page 497

For information about registering connection-level events, see:

- “ml_add_java_connection_script system procedure” on page 612
- “ml_add_dnet_connection_script system procedure” on page 610

**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:
  - “UploadData interface [MobiLink server Java]” on page 489
  - “UploadData interface [MobiLink server .NET]” on page 573

- **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:
  - “UploadedTableData interface [MobiLink server Java]” on page 491
  - “UploadedTableData interface [MobiLink server .NET]” on page 575

- **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 interface [MobiLink server Java]” on page 487.

  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 [MobiLink server .NET]” on page 572.

**Example**

See “handle_UploadData connection event” on page 352.

**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 “Conflict handling” on page 115.

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 database 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:

\[
\text{current data source} - (\text{old remote} - \text{new remote}) \\
\rightarrow 7 - (10-6) = 3
\]

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
  );

Java

1. Register a Java method for the handle_UploadData connection event.

   See “handle_UploadData connection event” on page 352.

   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.

   ```java
   call ml_add_java_connection_script( 'ver1',
        'handle_UploadData',
        'OrderProcessor.HandleUpload' )
   ```

   For more information about registering methods for synchronization events, see:

   o “Script additions and deletions” on page 235
   o “ml_add_java_connection_script system procedure” on page 612

2. Obtain an UpdateResultSet for a table in the upload.

   The OrderProcessor.HandleUpload method obtains an UpdateResultSet for the remoteOrders table:

   ```java
   // 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.

   ```java
   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.

```java
// 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.

```java
// 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);
}
```

.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.

```sql
call ml_add_dnet_connection_script( 'ver1',
    'handle_UploadData',
    'MyScripts.OrderProcessor.HandleUpload' )
```
For more information about registering methods for synchronization events, see:

- “Script additions and deletions” on page 235
- “ml_add_dnet_connection_script system procedure” on page 610

2. Obtain an UpdateDataReader for a table in the upload.

The MyScripts.OrderProcessor.HandleUpload method obtains an UpdateResultSet for the remoteOrders table:

```csharp
// 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.

```csharp
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.

```csharp
// 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.

```java
// 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);
}
```

### Direct downloads

To handle direct downloads:

1. Register a Java or .NET method for the handle_DownloadData connection event. See “handle_DownloadData connection event” on page 341.

2. Write a method for the handle_DownloadData synchronization event. In this event you use an instance of DBConnectionContext to get a DownloadData instance for the current synchronization. See:

   - “DBConnectionContext interface [MobiLink server Java]” on page 440
   - “DownloadData interface [MobiLink server Java]” on page 445
   - “DBConnectionContext interface [MobiLink server .NET]” on page 524
   - “DownloadData interface [MobiLink server .NET]” on page 545

   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 252.

### 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:
  - “DownloadData interface [MobiLink server Java]” on page 445
  - “DownloadData interface [MobiLink server .NET]” on page 545

- **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:
  - “DownloadTableData interface [MobiLink server Java]” on page 448
  - “DownloadTableData interface [MobiLink server .NET]” on page 547

**Example**

See “handle_DownloadData connection event” on page 341.
MobiLink reference

This section contains MobiLink reference material.

MobiLink Replay C++ callbacks

This section describes the public structure of the mlreplaycallbacks.cpp file that can be generated by the mlgenreplayapi utility. It contains a complete list of callbacks that you can develop to customize the data uploaded to the MobiLink server during a replay session using the mlreplay utility.

Note

If callbacks are not used, you must wait for all simulated clients to be ready to replay before they can start replaying. Simulated clients cannot perform the replay if any simulated client cannot be created or initialized successfully.

See also

● “MobiLink Replay utility (mlreplay)” on page 656

CreateAndInitMLReplayUploadTransaction callback

Used to create and initialize an upload transaction; called once initially, and then once per upload transaction, per synchronization, per simulated client, and per repetition for all repetitions greater than 1.

Syntax

```c
_MLREPLAY_EXPORT bool _MLREPLAY_CDECL CreateAndInitMLReplayUploadTransaction(
    IMLReplayUploadTransaction ** uploadTrans,
    const IMLReplayAPICallbacks * mlrAPICallbacks
)
```

Parameters

● uploadTrans An implementation of IMLReplayUploadTransaction that mlreplay uses to populate the replay session with custom data.

● mlrAPICallbacks Callbacks to provide information from mlreplay.

Returns

True on success; returns false on error, which cancels the replay session.

Remarks

You should not modify this callback.
DelayCreationOfSimulatedClient callback

Can be used to coordinate when each simulated client is created based on the given simulated client number and the number of simulated clients.

Syntax

```c
_HEREPLAY_EXPORT bool _HEREPLAY_CDECL DelayCreationOfSimulatedClient(
    asa_uint32 simulatedClientNum,
    const IMLReplayAPICallbacks * mlrAPICallbacks
)
```

Parameters

- `simulatedClientNum` The simulated client number (ordinal 1) used to distinguish this simulated client from other simulated clients in the same mlreplay instance.

- `mlrAPICallbacks` Callbacks to provide information from mlreplay that can be used to customize replay behavior

Returns

True if the specified simulated client is supposed to be created; returns false when the specified simulated client is not created.

Remarks

Simulated client \( X \) is created under the following conditions:

- DelayCreationOfSimulatedClient returned for simulated clients 1, ..., \( X - 1 \).
- DelayCreationOfSimulatedClient( \( X, mlrAPICallbacks \) ) returns true.

The simulated client is not created when this callback returns false and additional simulated clients are still created. This callback is called once per simulated client per mlreplay instance.

DelayDestructionOfSimulatedClient callback

Can be used to coordinate when each simulated client is destroyed; called once per simulated client, per mlreplay instance.

Syntax

```c
_HEREPLAY_EXPORT bool _HEREPLAY_CDECL DelayDestructionOfSimulatedClient(
    asa_uint32 simulatedClientNum,
    const IMLReplayAPICallbacks * mlrAPICallbacks
)
```

Parameters

- `simulatedClientNum` The simulated client number (ordinal 1) used to distinguish this simulated client from other simulated clients in the same mlreplay instance.


- **mlrAPICallbacks**
  
  Callbacks to provide information from mlreplay that can be used to customize replay behavior.

**Remarks**

This callback can be called concurrently.

Simulated client, X, won’t be destroyed until DelayDestructionOfSimulatedClient( X, mlrAPICallbacks ) returns.

---

### DelayStartOfReplay callback

Can be used to coordinate when replaying begins; called once per repetition, per simulated client per mlreplay instance.

**Syntax**

```cpp
bool DelayStartOfReplay(
    asa_uint32 repetitionNum,
    uint32 simulatedClientNum,
    const IMLReplayAPICallbacks * mlrAPICallbacks
)
```

**Parameters**

- **repetitionNum**
  
  The current repetition number.

- **simulatedClientNum**
  
  The simulated client number (ordinal 1) used to distinguish this simulated client from other simulated clients in the same mlreplay instance.

- **mlrAPICallbacks**
  
  Callbacks to provide information from mlreplay that can be used to customize replay behavior.

**Returns**

True to start replaying; false to skip the generation.

**Remarks**

This callback can be called concurrently.

If false is returned for repetition X, then the repetition is skipped.

---

### DestroyMLReplayUploadTransaction callback

Used to de-construct an upload transaction; called once initially after the first call to CreateAndInitMLReplayUploadTransaction, then once per upload transaction, per synchronization, per simulated client, and per repetition.
Syntax

```c
_MLREPLAY_EXPORT void _MLREPLAY_CDECL DestroyMLReplayUploadTransaction(
    IMLReplayUploadTransaction * uploadTrans
)
```

Parameters

- **uploadTrans**  An implementation of IMLReplayUploadTransaction that the mlreplay utility used to populate the replay session with custom data.

Remarks

You should not modify this callback.

**FinIdentifySimulatedClient callback**

Used to clean up memory used by the call to IdentifySimulatedClient for the given simulated client; called once per simulated client.

Syntax

```c
_MLREPLAY_EXPORT void _MLREPLAY_CDECL FiniIdentifySimulatedClient ( 
    asa_uint32 simulatedClientNum,
    char * remoteID,
    char * username,
    char * password,
    char * scriptVersion,
    char ** authenticationParameters,
    asa_uint16 numAuthenticationParameters,
    char * ldt,
    const IMLReplayAPICallbacks * mlrAPICallbacks
)
```

Parameters

- **simulatedClientNum**  The simulated client number (ordinal 1) used to distinguish this simulated client from other simulated clients in the same 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.

- **scriptVersion**  The script version 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 last download time given by the call to IdentifySimulatedClient for the given simulated client.

- **mlrAPICallbacks**  Callbacks to provide information from mlreplay that can be used to customize replay behavior.

**Remarks**

When generated, this callback contains commented out code that you can implement when using a generic username, password, and remote ID for a replay session. The code frees any used memory.

**GetDownloadApplyTime callback**

Used to simulate slow devices; called once per download, per simulated client, and per repetition.

**Syntax**

```c
_MLREPLAY_EXPORT asa_uint32 _MLREPLAY_CDECL GetDownloadApplyTime(
    asa_uint32 repetitionNum,
    asa_uint32 simulatedClientNum,
    asa_uint32 recordedSyncNum,
    asa_uint32 recordedDownloadApplyTime,
    const IMLReplayAPICallbacks * mlrAPICallbacks
)
```

**Parameters**

- **repetitionNum**  The current repetition number.

- **simulatedClientNum**  The simulated client number (ordinal 1) used to distinguish this simulated client from other simulated clients in the same mlreplay instance.

- **recordedSyncNum**  The synchronization number (ordinal 1) within the recorded protocol.

- **recordedDownloadApplyTime**  The recorded download apply time (in milliseconds).

- **mlrAPICallbacks**  Callbacks to provide information from mlreplay that can be used to customize replay behavior.

**Returns**

The number of milliseconds it should take to apply the download.

**Remarks**

This callback can be called concurrently.

The mlreplay utility 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.
GetMLReplayAPIVersion callback

Used to return the replay API version.

Syntax

```c
_MLREPLAY_EXPORT asa_uint32 _MLREPLAY_CDECL GetMLReplayAPIVersion( void )
```

Remarks

This callback is called once per mlreplay instance and should not be modified.

GetUploadTransaction callback

Used to customize the rows uploaded to the MobiLink server during the replay session; called once per upload transaction, per synchronization, per simulated client, and per repetition.

**Note**

The mlreplay utility tries to adjust the timing information based on the size of the new upload given by GetUploadTransaction and the upload in the recorded protocol file. However, if timing is important, best results are obtained if the total size of the rows added using GetUploadTransaction roughly match the size of the rows uploaded in the original recorded synchronization. The easiest way to ensure the size is roughly the same is to record a synchronization that uploads the same number of rows with similar data.

Syntax

```c
_MLREPLAY_EXPORT bool _MLREPLAY_CDECL GetUploadTransaction(
    asa_uint32 repetitionNum,
    asa_uint32 simulatedClientNum,
    asa_uint32 recordedSyncNum,
    asa_uint32 uploadTransNum,
    asa_uint32 numUploadedTrans,
    IMLReplayUploadTransaction * uploadTrans,
    const IMLReplayAPICallbacks * mlrAPICallbacks
)
```

Parameters

- **repetitionNum**  The current repetition number.
- **simulatedClientNum**  The simulated client number (ordinal 1) used to distinguish this simulated client from other simulated clients in the same mlreplay instance.
- **recordedSyncNum**  The synchronization number (ordinal 1) within the recorded protocol.
- **uploadTransNum**  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.
- **mlrAPICallbacks**  
  Callbacks to provide information from mlreplay that can be used to customize replay behavior.

**Returns**

Returns true on success; false on error. If GetUploadTransaction fails prior to the first repetition, then the replay session is canceled. If GetUploadTransaction fails prior to any repetition other than the first one, then only the failing simulated client is terminated.

**Remarks**

It may be called several times based on the number of synchronization and upload transactions that appear when the recorded protocol file is replayed.

This callback may be called concurrently but concurrent calls do not have a pointer to the same uploadTrans object.

---

**GlobalFini callback**

Used to clean up any global variables used by the other callbacks; called once per mlreplay instance.

**Syntax**

```c
_MLREPLAY_EXPORT void _MLREPLAY_CDECL GlobalFini(
    const IMLReplayAPICallbacks * mlrAPICallbacks
)
```

**Parameters**

- **mlrAPICallbacks**  
  Callbacks to provide information from mlreplay that can be used to customize replay behavior.

---

**GlobalInit callback**

Used to initialize any global variables used by the other callbacks; called once per mlreplay instance.

**Syntax**

```c
_MLREPLAY_EXPORT bool _MLREPLAY_CDECL GlobalInit(
    const IMLReplayAPICallbacks * mlrAPICallbacks
)
```

**Parameters**

- **mlrAPICallbacks**  
  Callbacks to provide information from mlreplay that can be used to customize replay behavior.

**Returns**

True on success; false on error, which cancels the replay session.
IdentifySimulatedClient callback

Used to specify the simulated client information; called once per simulated client.

Syntax

```c
_MLREPLAY_EXPORT bool _MLREPLAY_CDECL IdentifySimulatedClient(
    asa_uint32 simulatedClientNum,
    char ** remoteID,
    char ** username,
    char ** password,
    char ** scriptVersion,
    char *** authenticationParameters,
    asa_uint16 * numAuthenticationParameters,
    char ** ldt,
    const IMLReplayAPICallbacks * mlrAPICallbacks
)
```

Parameters

- **simulatedClientNum**  The simulated client number (ordinal 1) used to distinguish this simulated client from other simulated clients in the same 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.

- **scriptVersion**  An output parameter that must be set to the script version for the MobiLink user to use.

- **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 last download time for the user. The format of ldt must be yyyy-MM-dd hh:mm:ss.SSS.

- **mlrAPICallbacks**  Callbacks to provide information from mlreplay that can be used to customize replay behavior.

Returns

True on success, false on error, which cancels the replay session.

Remarks

If the username, password, authenticationParameters, scriptVersion, 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.
When generated, this callback contains commented out code that you can implement when using a generic username, password, and remote ID for a replay session. The code creates a username, password, and remote ID of `user_simulated client number`, `pwd_simulated client number`, and `rid_simulated client number`, respectively.

**ReportEndOfReplay callback**

Used to perform any actions required when a simulated client is finished replaying; called once per simulated client, and per repetition.

**Syntax**

```c
__MLREPLAY_EXPORT bool __MLREPLAY_CDECL ReportEndOfReplay (  
    asa_uint32 repetitionNum,  
    asa_uint32 simulatedClientNum,  
    bool success,  
    const IMLReplayAPICallbacks * mlrAPICallbacks
)
```

**Parameters**

- **repetitionNum**  The current repetition number.
- **simulatedClientNum**  The simulated client number (ordinal 1) used to distinguish this simulated client from other simulated clients in the same mlreplay instance.
- **success**  True when the simulated client completed successfully; otherwise, false.
- **mlrAPICallbacks**  Callbacks to provide information from mlreplay that can be used to customize replay behavior.

**Returns**

True on success; false on error.

**Remarks**

You can use this callback to ensure that data was uploaded correctly in repetition X.

This callback determines the result of the given replay. If success is false, then this callback has no affect on the success of the replay. If success is true but this callback returns false, then mlreplay treats the simulated client as if it failed for the specified repetition. This callback can be called concurrently.

**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”
- “ml_add_table_script system procedure”
- “ml_add_dnet_connection_script system procedure”
- “ml_add_dnet_table_script system procedure”
- “ml_add_java_connection_script system procedure”
- “ml_add_java_table_script system procedure”

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:

- “ml_ra_add_agent_id system procedure”
- “ml_ra_assign_task system procedure”
- “ml_ra_cancel_notification system procedure”
- “ml_ra_cancel_task_instance system procedure”
- “ml_ra_clone_agent_properties system procedure”
- “ml_ra_delete_agent_id system procedure”
- “ml_ra_delete_events_before system procedure”
- “ml_ra_delete_remote_id system procedure”
- “ml_ra_delete_task system procedure”
- “ml_ra_get_agent_events system procedure”
- “ml_ra_get_agent_ids system procedure”
- “ml_ra_get_agent_properties system procedure”
- “ml_ra_get_latest_event_id system procedure”
- “ml_ra_get_orphan_taskdbs system procedure”
- “ml_ra_reassign_taskdb system procedure”
- “ml_ra_get_remote_ids system procedure”
- “ml_ra_get_task_results system procedure”
- “ml_ra_get_task_status system procedure”
- “ml_ra_manage_remote_db system procedure”
- “ml_ra_notify_agent_sync system procedure”
- “ml_ra_set_agent_property system procedure”
- “ml_ra_unmanage_remote_db system procedure”

LDAP system procedures

The following stored procedures can be used to setup and manage LDAP authentication:

- “ml_add_certificates_file system procedure” on page 608
- “ml_add_ldap_server system procedure” on page 615
- “ml_add_user_auth_policy system procedure” on page 626

Synchronization model system procedures

The following stored procedures can be used to manage schema upgrades:

- “ml_model_drop system procedure” on page 631
- “ml_model_check_all_schema system procedure” on page 632
- “ml_model_check_version_schema system procedure” on page 633

Other system procedures

- “ml_add_property system procedure”
- “ml_delete_sync_state_before system procedure”
- “ml_reset_sync_state system procedure”
ml_add_certificates_file system procedure

Set up trusted certificates when using TLS with LDAP authentication.

Syntax

```
ml_add_certificates_file (  
  'file_name',  
)
```

Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>file_name</td>
<td>VARCHAR(1024). The name of the trusted certificates file.</td>
</tr>
</tbody>
</table>

Remarks

This procedure populates the ml_trusted_certificates_file table with information about the specified trusted certificate file.

Existing entries in the ml_trusted_certificates_file table are deleted before the new trusted certificate file name is inserted because only a single trusted certificate file is required for a server farm.

See also

- “ml_add_ldap_server system procedure” on page 615
- “ml_add_user_auth_policy system procedure” on page 626

ml_add_column system procedure (deprecated)

Registers information about columns on remote databases for use by named column parameters.

Syntax

```
ml_add_column (  
  'version',  
  'table',  
  'column',  
  'type'  
)
```

Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>VARCHAR(128). The version name.</td>
</tr>
<tr>
<td>table</td>
<td>VARCHAR(128). The table name.</td>
</tr>
<tr>
<td>column</td>
<td>VARCHAR(128). The column name.</td>
</tr>
</tbody>
</table>
### ml_add_column system procedure

Use this system procedure to populate the ml_column MobiLink system table with information about the columns on the remote database. The information is used by named row parameters.

**Syntax**

```
ml_add_column( 'version', 'table_name', 'column_name', null )
```

**Description**

- **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 and later clients do send up column names, so ml_add_column is not required in most deployments. The ml_add_column names always override names from the client.

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 216

### 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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>VARCHAR(128). The version name.</td>
</tr>
<tr>
<td>event</td>
<td>VARCHAR(128). The event name.</td>
</tr>
<tr>
<td>script</td>
<td>TEXT. The script contents. For Adaptive Server Enterprise, this parameter is VARCHAR(16384). For IBM DB2 LUW, this parameter is VARCHAR(4000). For SAP HANA and Oracle, this parameter is CLOB.</td>
</tr>
</tbody>
</table>

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 606
- “Script additions and deletions” on page 235
- “ml_add_table_script system procedure” on page 624
- “ml_add_dnet_connection_script system procedure” on page 610
- “ml_add_dnet_table_script system procedure” on page 611
- “ml_add_java_connection_script system procedure” on page 612
- “ml_add_java_table_script system procedure” on page 614

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.

```sql
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

```sql
ml_add_dnet_connection_script ( 'version',
    'event',
    'script'
)
```
Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>VARCHAR(128). The version name.</td>
</tr>
<tr>
<td>event</td>
<td>VARCHAR(128). The event name.</td>
</tr>
<tr>
<td>script</td>
<td>TEXT. The script contents. For Adaptive Server Enterprise, this parameter is</td>
</tr>
<tr>
<td></td>
<td>VARCHAR(16384). For IBM DB2 LUW, this parameter is VARCHAR(4000). For SAP</td>
</tr>
<tr>
<td></td>
<td>HANA and Oracle, this parameter is CLOB.</td>
</tr>
</tbody>
</table>

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 606
- “Script additions and deletions” on page 235
- “ml_add_dnet_table_script system procedure” on page 611
- “ml_add_connection_script system procedure” on page 609
- “ml_add_table_script system procedure” on page 624
- “ml_add_java_table_script system procedure” on page 614
- “Methods” on page 502
- “Synchronization scripts in .NET” on page 497

Example

The following example registers the beginDownloadConnection method of the ExampleClass class for the begin_download event.

```sql
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

```sql
ml_add_dnet_table_script (  
    'version',  
    'table',  
    'event',  
```
'script'
)

Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>VARCHAR(128). The version name.</td>
</tr>
<tr>
<td>table</td>
<td>VARCHAR(128). The table name.</td>
</tr>
<tr>
<td>event</td>
<td>VARCHAR(128). The event name.</td>
</tr>
<tr>
<td>script</td>
<td>TEXT. The script contents. For Adaptive Server Enterprise, this parameter is VARCHAR(16384). For IBM DB2 LUW, this parameter is VARCHAR(4000). For SAP HANA and Oracle, this parameter is CLOB.</td>
</tr>
</tbody>
</table>

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 606
- “Script additions and deletions” on page 235
- “ml_add_dnet_connection_script system procedure” on page 610
- “ml_add_connection_script system procedure” on page 609
- “ml_add_table_script system procedure” on page 624
- “ml_add_java_connection_script system procedure” on page 612
- “Methods” on page 502
- “Synchronization scripts in .NET” on page 497

Example

The following example assigns the `empDownloadCursor` method of the `EgClass` class to the `download_cursor` event for the table `emp`.

```sql
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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>VARCHAR(128). The version name.</td>
</tr>
<tr>
<td>event</td>
<td>VARCHAR(128). The event name.</td>
</tr>
<tr>
<td>script</td>
<td>TEXT. The script contents. For Adaptive Server Enterprise, this parameter is VARCHAR(16384). For IBM DB2 LUW, this parameter is VARCHAR(4000). For SAP HANA and Oracle, this parameter is CLOB.</td>
</tr>
</tbody>
</table>

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 606
- “Script additions and deletions” on page 235
- “ml_add_connection_script system procedure” on page 609
- “ml_add_table_script system procedure” on page 624
- “ml_add_dnet_connection_script system procedure” on page 610
- “ml_add_dnet_table_script system procedure” on page 611
- “ml_add_java_table_script system procedure” on page 614
- “Methods” on page 431
- “Synchronization script writing in Java” on page 427

Example

The following example registers the endConnection method of the CustEmpScripts class for the end_connection event.

```sql
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

```sql
ml_add_java_table_script(
  'version',
  'table',
  'event',
  'script'
)
```

Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>VARCHAR(128). The version name.</td>
</tr>
<tr>
<td>table</td>
<td>VARCHAR(128). table name.</td>
</tr>
<tr>
<td>event</td>
<td>VARCHAR(128). The event name.</td>
</tr>
<tr>
<td>script</td>
<td>TEXT. The script content. For Adaptive Server Enterprise, this parameter is VARCHAR(16384). For IBM DB2 LUW, this parameter is VARCHAR(4000). For SAP HANA and Oracle, this parameter is CLOB.</td>
</tr>
</tbody>
</table>

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 606
- “Script additions and deletions” on page 235
- “ml_add_connection_script system procedure” on page 609
- “ml_add_table_script system procedure” on page 624
- “ml_add_dnet_connection_script system procedure” on page 610
- “ml_add_dnet_table_script system procedure” on page 611
- “ml_add_java_connection_script system procedure” on page 612
- “Methods” on page 431
- “Synchronization script writing in Java” on page 427

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.

**See also**

- “ml_add_connection_script system procedure” on page 609
- “ml_add_dnet_connection_script system procedure” on page 610
- “ml_add_java_connection_script system procedure” on page 612

**ml_add_lang_connection_script_chk system procedure**

This procedure is for internal use only.

**See also**

- “ml_add_connection_script system procedure” on page 609
- “ml_add_dnet_connection_script system procedure” on page 610
- “ml_add_java_connection_script system procedure” on page 612

**ml_add_lang_table_script system procedure**

This procedure is for internal use only.

**See also**

- “ml_add_table_script system procedure” on page 624
- “ml_add_java_table_script system procedure” on page 614
- “ml_add_dnet_table_script system procedure” on page 611

**ml_add_lang_table_script_chk system procedure**

This procedure is for internal use only.

**See also**

- “ml_add_table_script system procedure” on page 624
- “ml_add_java_table_script system procedure” on page 614
- “ml_add_dnet_table_script system procedure” on page 611

**ml_add_ldap_server system procedure**

Creates, drops or updates LDAP servers.
Syntax

```sql
ml_add_ldap_server (  'ldsrv_name',  'search_url',  'access_dn',  'access_dn_pwd'  'auth_url'  'conn_retries'  'conn_timeout'  'use_tls'
)
```

Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ldsrv_name</td>
<td>VARCHAR(128). A unique LDAP server name.</td>
</tr>
<tr>
<td>search_url</td>
<td>VARCHAR(1024). A URL string that identifies the host by name or IP address,</td>
</tr>
<tr>
<td></td>
<td>port number, and search string to perform the DN (distinguished name) lookup</td>
</tr>
<tr>
<td></td>
<td>for a given user id.</td>
</tr>
<tr>
<td>access_dn</td>
<td>VARCHAR(1024). The distinguished name for an LDAP user that is used by the</td>
</tr>
<tr>
<td></td>
<td>MobiLink server to connect to the LDAP server. The LDAP user must have per-</td>
</tr>
<tr>
<td></td>
<td>mission on the LDAP server to search for DNs.</td>
</tr>
<tr>
<td>access_dn_pwd</td>
<td>VARCHAR(1024). The password associated with the DN specified with the ac-</td>
</tr>
<tr>
<td></td>
<td>cess_dn parameter.</td>
</tr>
<tr>
<td>auth_url</td>
<td>VARCHAR(1024). A URL string that identifies the host by name or IP address</td>
</tr>
<tr>
<td></td>
<td>and the port number of the LDAP server used to authenticate a user.</td>
</tr>
<tr>
<td>conn_retries</td>
<td>TINYINT. The number of times the MobiLink server tries to connect to the</td>
</tr>
<tr>
<td></td>
<td>LDAP server for DN searches and authentication. The valid range is 1-60.</td>
</tr>
<tr>
<td></td>
<td>The default is 3.</td>
</tr>
<tr>
<td>conn_timeout</td>
<td>TINYINT. The connection timeout from the MobiLink server to the LDAP server</td>
</tr>
<tr>
<td></td>
<td>for DN searches and authentication. The value is specified in seconds.</td>
</tr>
<tr>
<td></td>
<td>The default value is 10 seconds.</td>
</tr>
<tr>
<td>start_tls</td>
<td>TINYINT. Specifies that TLS be used for connections to the LDAP server for</td>
</tr>
<tr>
<td></td>
<td>DN searches and authentication.</td>
</tr>
</tbody>
</table>

Remarks

This procedure populates the ml_ldap_server table with information about the specified LDAP server.

See also

- “ml_add_certificates_file system procedure” on page 608
- “ml_add_user_auth_policy system procedure” on page 626
Example
The following example adds an LDAP server named **my_primary** into the ml_ldap_server table.

```sql
CALL ml_add_ldap_server(
    'my_primary',                                       //server name
    'ldap://voyager:389/dc=MyCompany,dc=com??sub?cn=*',  //search URL
    'cn=aseadmin, cn=Users, dc=mycompany, dc=com',       //access DN
    'Secret99Password',                                  //access DN password
    'ldap://voyager:389/',                               //authentication URL
    10,                                                  //connection retries
    5,                                                   //connection timeout
    0                                                    //no TLS
);
```

**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**

```sql
ml_add_missing_dnld_scripts (  
    'script_version_name')
```

**Parameters**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>script_version_name</td>
<td>VARCHAR(128). The name of the script version.</td>
</tr>
</tbody>
</table>

**See also**

- “download_cursor table event” on page 301
- “download_delete_cursor table event” on page 303

**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**

```sql
ml_add_passthrough (  
    'remote_id',  
    'script_name',  
    run_order  
)
```
### Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remote_id</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td><strong>Caution</strong></td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>script_name</td>
<td>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.</td>
</tr>
<tr>
<td>run_order</td>
<td>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 database 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.</td>
</tr>
<tr>
<td></td>
<td>This value must be a non-negative integer or null.</td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>failed_script_name</td>
<td>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.</td>
</tr>
<tr>
<td>error_code</td>
<td>INTEGER. The SQL Anywhere error code that this rule handles.</td>
</tr>
<tr>
<td>new_script_name</td>
<td>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.</td>
</tr>
<tr>
<td>action</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>● 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.</td>
</tr>
<tr>
<td></td>
<td>● P  (purge) Indicates that the remote database should discard all the scripts that it has received and continue executing script normally after that.</td>
</tr>
<tr>
<td></td>
<td>● S  (skip) Indicates that the remote database should ignore the failed script and continue executing scripts as if the failed script had succeeded.</td>
</tr>
<tr>
<td></td>
<td>● H  (halt) Indicates that the remote database should not execute any more scripts until it receives further instructions.</td>
</tr>
</tbody>
</table>

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

```sql
ml_add_passthrough_script ( 
  'script_name', 
  'flags', 
  'affected_pubs', 
  'script', 
  'description' 
)
```
### Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>script_name</td>
<td>VARCHAR(128). The script name. This value must be unique.</td>
</tr>
<tr>
<td>flags</td>
<td>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:</td>
</tr>
<tr>
<td></td>
<td>● <strong>manual</strong> 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.</td>
</tr>
<tr>
<td></td>
<td>● <strong>exclusive</strong> 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.</td>
</tr>
<tr>
<td></td>
<td>● <strong>schema_diff</strong> 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.</td>
</tr>
<tr>
<td>affected_pubs</td>
<td>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.</td>
</tr>
</tbody>
</table>

For example:

'`manual;exclusive;schema_diff'`
**Syntax**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>script</td>
<td>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 <code>go</code>. The word <code>go</code> 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 <code>BEGIN...END</code> block.</td>
</tr>
<tr>
<td>description</td>
<td>VARCHAR(2000). A comment or description of the script. This value may be null.</td>
</tr>
</tbody>
</table>

**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**

```sql
ml_add_property ( 'comp_name', 'prop_set_name', 'prop_name', 'prop_value' )
```
Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>comp_name</td>
<td>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.</td>
</tr>
<tr>
<td>prop_set_name</td>
<td>VARCHAR(128). The property set name.</td>
</tr>
<tr>
<td></td>
<td>If the component name is ScriptVersion, then this parameter is the name of the script version.</td>
</tr>
<tr>
<td></td>
<td>If the component name is MLS, then this parameter can be ml_user_logverbosity to specify verbosity for a MobiLink user, ml_remote_id_logverbosity to specify verbosity for a remote ID, or locking_and_blocking_detection to report locking and blocking information to the MobiLink Profiler or MobiLink server log file.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>prop_name</td>
<td>VARCHAR(128). The property name.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>If the component name is MLS, then this property is either a MobiLink user name or remote ID that you define, or blocking_threshold_in_seconds for reporting locking and blocking information to the MobiLink Profiler or MobiLink server log file.</td>
</tr>
<tr>
<td>prop_value</td>
<td>TEXT. The property value.</td>
</tr>
<tr>
<td></td>
<td>If the prop_set_name is ml_user_logverbosity or ml_remote_id_logverbosity, this must be a valid mlsrv -v option.</td>
</tr>
<tr>
<td></td>
<td>For reporting locking and blocking information to the MobiLink Profiler or MobiLink server log file, this value is time_in_seconds.</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
</tbody>
</table>

Locking and blocking reporting

The MobiLink server detects any user-defined scripts running longer than a certain time (the default value is 60 seconds) and then reports the locking/blocking information to the MobiLink Profiler, if it is connected to the MobiLink server, and also logs the information into the MobiLink server log file.
The locking/blocking information includes the following:

- the synchronization ID
- the MobiLink server connection ID that is currently blocked
- the connection ID that is currently blocking the MobiLink server connection
- the total blocked time in seconds
- the object or operation name that the server connection is blocked on

The default time can be changed by executing the following SQL statement on the consolidated database:

```sql
call ml_add_property( 'MLS', 'locking_and_blocking_detection', 'blocking_threshold_in_seconds', 'time_in_seconds' );
```

where `time_in_seconds` is an integer that gives the blocking threshold in seconds. When `time_in_seconds` is zero, this feature is disabled.

This is a static property. The MobiLink server must be restarted before the new values take effect.

**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 the MobiLink server 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 execute the following SQL statement:

```sql
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 execute the following SQL statement:

```sql
call ml_add_property( 'MLS', 'ml_remote_id_log_verbosity', 'rid_1', '-v+' )
```

The `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 uses this `verbose_setting` for `ml_user1` or `rid_1` after 5 minutes. See “-v mlsrv16 option” on page 79.

To disable log verbosity for a MobiLink user, log into the consolidated database and execute the following SQL statement:

```sql
call ml_add_property( 'MLS', 'ml_user_log_verbosity', 'ml_user1', NULL )
```

To disable log verbosity for a MobiLink remote ID, log into the consolidated database and execute the following SQL statement:

```sql
call ml_add_property( 'MLS', 'ml_remote_id_log_verbosity', 'rid_1', NULL )
```
The MobiLink server stops using the previous verbose setting for `ml_user1` 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 uses 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`, execute the following SQL statement:

```
ml_add_property( 'SIS','SMTP(x)','server','mailserver1' );
```

The verbosity property applies to all Notifiers and gateways so you cannot specify a particular property set name. To change the verbosity setting, leave the property set name blank. For example:

```
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`, execute the following SQL statement:

```
ml_add_property( 'ScriptVersion','MyVersion','ldap-server','MyServer' );
```

See also

- “MobiLink server settings for server-initiated synchronization” [MobiLink - Server-Initiated Synchronization]
- “DBConnectionContext.getProperties method [MobiLink server Java]” on page 443
- “DBConnectionContext.get_set_name method [MobiLink server Java]” on page 445
- “DBConnectionContext.getProperties method [MobiLink server .NET]” on page 527 and “DBConnectionContext.get_set_name method [MobiLink server .NET]” on page 529
- “ServerContext.getPropertiesByVersion method [MobiLink server Java]” on page 472
- “ServerContext.getProperties method [MobiLink server Java]” on page 471
- “ServerContext.getPropertySetNames method [MobiLink server Java]” on page 472

**ml_add_table_script system procedure**

Use this system procedure to add or delete SQL table scripts in the consolidated database.
Syntax

\[
\text{ml\_add\_table\_script (}
\begin{array}{l}
'\text{version}',
'table',
'\text{event}',
'script'
\end{array}
\)
\]

Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>VARCHAR(128). The version name.</td>
</tr>
<tr>
<td>table</td>
<td>VARCHAR(128). The table name.</td>
</tr>
<tr>
<td>event</td>
<td>VARCHAR(128). The event name.</td>
</tr>
<tr>
<td>script</td>
<td>TEXT. The script contents. For Adaptive Server Enterprise, this parameter is VARCHAR(16384). For IBM DB2 LUW, this parameter is VARCHAR(4000). For SAP HANA and Oracle, this parameter is CLOB.</td>
</tr>
</tbody>
</table>

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.

The MobiLink server needs to be restarted for the specified script changes to take effect, unless the MobiLink server was started with the -zf mlsrv16 option. The -zf option causes the MobiLink server to check for script changes at the beginning of each synchronization.

Caution

Running the MobiLink server with the -zf option has a negative impact on MobiLink server performance and should be avoided whenever possible.

See also

- “System procedures to add or delete scripts” on page 606
- “Script additions and deletions” on page 235
- “ml_add_connection_script system procedure” on page 609
- “ml_add_dnet_connection_script system procedure” on page 610
- “ml_add_dnet_table_script system procedure” on page 611
- “ml_add_java_connection_script system procedure” on page 612
- “ml_add_java_table_script system procedure” on page 614
- “-zf mlsrv16 option” on page 94
Example

The following command adds a table script associated with the upload_insert event on the Customer table.

```sql
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_add_user_auth_policy** system procedure

Use this system procedure to add MobiLink user authentication policies.

**Syntax**

```
ml_add_user_auth_policy ( 'policy_name', 'primary_ldsrv_name', 'secondary_ldsrv_name', 'ldap_auto_failback_period', 'ldap_failover_to_std' )
```

**Parameters**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>policy_name</code></td>
<td>VARCHAR(128). A unique user authentication policy name.</td>
</tr>
<tr>
<td><code>primary_ldsrv_name</code></td>
<td>VARCHAR(128). Specifies the primary LDAP server name to be used to authenticate this user. The specified LDAP server name must already exist in the ml_ldap_server table.</td>
</tr>
<tr>
<td><code>secondary_ldsrv_name</code></td>
<td>VARCHAR(128). Specifies the secondary LDAP server name for failover purposes. The secondary LDAP server name must already exist in the ml_ldap_server table.</td>
</tr>
<tr>
<td><code>ldap_auto_failback_period</code></td>
<td>INTEGER. Use this parameter to inform the MobiLink server when it should failover to the primary LDAP server for user authentication. The time is specified in seconds and the default value is 900 seconds (15 minutes).</td>
</tr>
<tr>
<td><code>ldap_failover_to_std</code></td>
<td>When the primary LDAP server is not available for user authentication, the MobiLink server remembers when the problem was detected and switches to the secondary server for user authentication. The MobiLink server then switch back to use the primary server for user authentication for whatever users who are currently using this user authentication policy, when the elapsed time since the failure was detected has reached @ldap_auto_failback_period.</td>
</tr>
</tbody>
</table>
**ldap_failover_to_std**

INTEGER. Specifies whether or not the MobiLink server should use standard methods (password and user authentication scripts) to authenticate the user when neither the primary or secondary LDAP server is available. The value can be 1 (TRUE) or 0 (FALSE). The default is 1.

**Remarks**

This procedure adds a user authentication policy to the ml_user_auth_policy table, if the specified policy_name does not exist in the table. If the policy_name is already in the table, executing this procedure with non-NULL parameters updates all the corresponding fields with the specified non-NULL parameters. For instance, the following SQL statement updates the user authentication policy policy_1 to use ldap_server2 as a secondary LDAP server and enables failover to use standard password and user authentication script based authentication, when both the primary and secondary LDAP servers are unavailable.

```
   call ml_add_user_auth_policy( 'policy_1', NULL, 'ldap_server2', NULL, 1 )
```

To delete an authentication policy, all parameters except for policy_name should be NULL.

When adding a MobiLink user authentication policy, the parameter primary_ldsrv_name cannot be NULL but the secondary_ldsrv_name parameter can be NULL.

**See also**

- “ml_add_ldap_server system procedure” on page 615
- “ml_add_certificates_file system procedure” on page 608

**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 database with the specified run order. If the script is downloaded to the remote database before it is deleted then it is not deleted from the remote database and executes as usual.

**Syntax**

```
   ml_delete_passthrough(  
      'remote_id',  
      'script_name',  
      'run_order'  
   )
```

**Parameters**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remote_id</td>
<td>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.</td>
</tr>
</tbody>
</table>
### Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>script_name</td>
<td>VARCHAR(128). The script name.</td>
</tr>
<tr>
<td>run_order</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>failed_script_name</td>
<td>VARCHAR(128). The name of the script to which a rule applied.</td>
</tr>
<tr>
<td>error_code</td>
<td>INTEGER. The error code for which the rule applied.</td>
</tr>
</tbody>
</table>

### 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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>script_name</td>
<td>VARCHAR(128). The name of the script to remove.</td>
</tr>
</tbody>
</table>

Remarks

Scripts cannot 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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>VARCHAR(128). The MobiLink user name.</td>
</tr>
<tr>
<td>remote_id</td>
<td>VARCHAR(128). The remote ID.</td>
</tr>
</tbody>
</table>

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**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ts</td>
<td>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.</td>
</tr>
</tbody>
</table>

**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 database 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_model_drop system procedure**

Use this procedure to drop synchronization models installed using the MobiLink 16 plug-in for Sybase Central.

**Syntax**

```
ml_model_drop ( 'script_version'
)
```

**Parameters**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>script_version</code></td>
<td>VARCHAR(128). The name of the script version associated with the synchronization model you want to drop.</td>
</tr>
</tbody>
</table>

**Remarks**

This stored procedure removes the synchronization scripts included in the named script version, as well as any schema that was created when the synchronization model was deployed, including shadow tables, tracking columns, triggers and indexes.

Schema that is shared with another `script_version` is not deleted.

No schema is deleted if a `script_version` was installed manually outside of the MobiLink 16 plug-in.

**See also**

- “ml_model_check_all_schema system procedure” on page 632
- “ml_model_check_version_schema system procedure” on page 633
ml_model_check_all_schema system procedure

Use this procedure to check the status of each schema object required by deployed synchronization models. This stored procedure returns information for all deployed synchronization models.

Syntax

ml_model_check_all_schema

Remarks

This procedure returns a result set containing the status of each schema object required by all deployed synchronization models.

No results are returned for script versions installed outside of Sybase Central or for synchronization models deployed prior to version 16.

The result set contains the following columns:

- **schema_owner** Identifies the schema owner.
- **table_name** Identifies the table name.
- **schema_type** Identifies the schema type. It can be one of the following types:
  - TABLE
  - INDEX
  - COLUMN
  - TRIGGER
  - PROCEDURE
- **object_name** Identifies the object name.
- **locked** If this column is set to 1 then the schema is never modified or dropped by the plug-in. Schema used by synchronization models that pre-existed before a deployment is marked as locked.
- **used_by** The script version that requires the schema object.
- **status** Status can be one of the following:
  - INSTALLED The schema is installed correctly.
  - MISSING The schema is not installed.
  - MISMATCH The installed schema is different from what is required.
  - UNVERIFIED The schema exists, but there is not enough information to determine that it is defined correctly.
  - UNUSED No synchronization model is using this schema object.
- **overwrite_action** Can be one of the following:
REPLACE  If the model is redeployed, the existing schema is dropped and recreated

CREATE  If the model is redeployed, the schema is created.

SKIP  Either the schema is already correctly installed, or there is conflicting schema that is blocking proper installation.

● preserve_action  Reserved for future use.

See also
● “ml_model_drop system procedure” on page 631
● “ml_model_check_version_schema system procedure” on page 633

ml_model_check_version_schema system procedure

Use this procedure to check the status of each schema object required by deployed synchronization models. This stored procedure returns information for the specified script version.

Syntax

ml_model_check_version_schema ( 'script_version'
)

Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>script_version</td>
<td>VARCHAR(128). The name of the script version associated with the synchronization model you want to check.</td>
</tr>
</tbody>
</table>

Remarks

This procedure returns a result set containing the status of each schema object required by the specified script version.

No results are returned for script versions installed outside of Sybase Central or for synchronization models deployed prior to version 16.

The result set contains the following columns:

● schema_owner  Identifies the schema owner.

● table_name  Identifies the table name.
- **schema_type** Identifies the schema type. It can be one of the following types:
  - TABLE
  - INDEX
  - COLUMN
  - TRIGGER
  - PROCEDURE

- **object_name** Identifies the object name.

- **locked** If this column is set to 1 then the schema is never modified or dropped by the plug-in. Schema used by synchronization models that pre-existed before a deployment is marked as locked.

- **status** Status can be one of the following:
  - INSTALLED The schema is installed correctly.
  - MISSING The schema is not installed.
  - MISMATCH The installed schema is different from what is required.
  - UNVERIFIED The schema exists, but there is not enough information to determine that it is defined correctly.
  - UNUSED No synchronization model is using this schema object.

- **overwrite_action** Can be one of the following:
  - REPLACE If the model is redeployed, the existing schema is dropped and recreated
  - CREATE If the model is redeployed, the schema is created.
  - SKIP Either the schema is already correctly installed, or there is conflicting schema that is blocking proper installation.

- **preserve_action** Reserved for future use.

See also
- “ml_model_drop system procedure” on page 631
- “ml_model_check_all_schema system procedure” on page 632

**ml_ra_add_agent_id system procedure**

Use this procedure to define a new remote agent in the consolidated database.
### ml_ra_add_agent_id system procedure

This procedure is used to add a new agent to the consolidated database.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>agent_id</strong></td>
<td>VARCHAR(128). This is an IN parameter that specifies the ID of the new agent to be defined in the consolidated database.</td>
<td></td>
</tr>
</tbody>
</table>

### 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 648
- “ml_ra_clone_agent_properties system procedure” on page 637
- “ml_ra_set_agent_property system procedure” on page 649

### ml_ra_assign_task system procedure

Use this procedure to assign a task to a specific remote agent.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>agent_id</strong></td>
<td>VARCHAR(128). This is an IN parameter that specifies the ID of the agent to assign the task to.</td>
<td></td>
</tr>
<tr>
<td><strong>task_name</strong></td>
<td>VARCHAR(128). This is an IN parameter that specifies the name of the specific task being assigned.</td>
<td></td>
</tr>
</tbody>
</table>

### Remarks

Tasks must first be defined in Sybase Central using the MobiLink 16 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 636
- “ml_ra_notify_task system procedure” on page 649
- “ml_ra_cancel_notification system procedure” on page 636

**ml_ra_cancel_notification system procedure**

Use this procedure to cancel a server initiated remote task (SIRT) request that is no longer needed.

**Parameters**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_id</td>
<td>VARCHAR(128). This is an IN parameter that specifies the ID of the agent responsible for the task you are canceling.</td>
</tr>
<tr>
<td>task_name</td>
<td>VARCHAR(128). This is an IN parameter that specifies the name of the specific task being canceled.</td>
</tr>
</tbody>
</table>

See also

- “ml_ra_notify_task system procedure” on page 649
- “ml_ra_assign_task system procedure” on page 635
- “ml_ra_delete_task system procedure” on page 639
- “ml(ra_cancel_task_instance system procedure” on page 636

**ml_ra_cancel_task_instance system procedure**

Use this procedure to cancel a remote task instance.

**Parameters**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_id</td>
<td>VARCHAR(128). This is an IN parameter that specifies the ID of the agent responsible for the task you are canceling.</td>
</tr>
<tr>
<td>task_name</td>
<td>VARCHAR(128). This is an IN parameter that specifies the name of the specific task being canceled.</td>
</tr>
</tbody>
</table>

**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.
ml_ra_clone_agent_properties system procedure

Use this procedure to set all remote agent properties at once.

Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst_agent_id</td>
<td>VARCHAR(128). This is an IN parameter that specifies the ID of the destination agent being created.</td>
</tr>
<tr>
<td>src_agent_id</td>
<td>VARCHAR(128). This is an IN parameter that specifies the ID of the agent being cloned to create the new agent.</td>
</tr>
</tbody>
</table>

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 649
- “ml_ra_add_agent_id system procedure” on page 634
- “ml_ra_manage_remote_db system procedure” on page 648

ml_ra_delete_agent_id system procedure

Use this procedure to delete a defined agent from the consolidated database.

Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_id</td>
<td>VARCHAR(128). This is an IN parameter that specifies the ID of the agent being deleted.</td>
</tr>
</tbody>
</table>
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 638
- “ml_ra_delete_remote_id system procedure” on page 638
- “ml_ra_unmanage_remote_db system procedure” on page 650

ml_ra_delete_events_before system procedure
Use this procedure to delete events that are no longer needed from the consolidated database.

Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete_rows_older_than</td>
<td>TIMESTAMP. This is an IN parameter that specifies Events older than the specified value are deleted from the consolidated database.</td>
</tr>
</tbody>
</table>

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 643

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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remote_id</td>
<td>VARCHAR(128). This is an IN parameter that specifies the remote ID that corresponds to the remote database to be deleted.</td>
</tr>
</tbody>
</table>

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.
ml_ra_delete_task system procedure

Use this procedure to delete a remote task from the consolidated database.

Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>VARCHAR(128). This is an IN parameter that specifies the name of the remote task to be deleted.</td>
</tr>
</tbody>
</table>

Remarks

This system procedure fails if there are still active task instances.

See also

- “ml_ra_delete_agent_id system procedure” on page 637
- “ml_ra_delete_task system procedure” on page 639
- “ml_ra_delete_events_before system procedure” on page 638
- “ml_ra_delete_remote_id system procedure” on page 638

ml_ra_get_agent_events system procedure

Use this procedure to query events.

Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start_at_event_id</td>
<td>BIGINT. This is an IN parameter that specifies the ID of the event from which to start the query.</td>
</tr>
<tr>
<td>max_events_to_fetch</td>
<td>BIGINT. This is an IN parameter that specifies the maximum number of events to fetch.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event_id</td>
<td>BIGINT. A unique ID assigned to each event. The value is incremented by 1 for each new event.</td>
</tr>
<tr>
<td>Result</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>event_class</td>
<td>VARCHAR(1). The event class. The class can be either I for information or E for error.</td>
</tr>
<tr>
<td>event_type</td>
<td>VARCHAR(8). The event types are listed below.</td>
</tr>
<tr>
<td></td>
<td><strong>ANEW</strong> A new agent was defined in the consolidated database. This can occur by calling ml_ra_add_agent or if the agent is not preconfigured when the agent connects to the consolidated database for the first time.</td>
</tr>
<tr>
<td></td>
<td><strong>AFIRST</strong> Occurs on the first synchronization of an agent.</td>
</tr>
<tr>
<td></td>
<td><strong>ADUP</strong> 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.</td>
</tr>
<tr>
<td></td>
<td><strong>ARESET</strong> An agent rebuilt its agent database. Some task progress and results may have been lost.</td>
</tr>
<tr>
<td></td>
<td><strong>TB</strong> A task has begun execution.</td>
</tr>
<tr>
<td></td>
<td><strong>TE</strong> The task execution ended without a fatal error.</td>
</tr>
<tr>
<td></td>
<td><strong>TW</strong> A task execution is waiting for a retry interval before continuing.</td>
</tr>
<tr>
<td></td>
<td><strong>TAC</strong> Task execution ended because a command aborted.</td>
</tr>
<tr>
<td></td>
<td><strong>TAT</strong> Task execution ended because it exceeded the maximum allowed running time.</td>
</tr>
<tr>
<td></td>
<td><strong>TAR</strong> Task execution ended because it exceeded the maximum retry count.</td>
</tr>
<tr>
<td></td>
<td><strong>TFS</strong> The task completed and will not run again because it was a run-once task that succeeded.</td>
</tr>
<tr>
<td></td>
<td><strong>TFF</strong> The task completed and will not run again because it was a run-once task that failed.</td>
</tr>
<tr>
<td></td>
<td><strong>TFE</strong> The task completed and will not run again because the schedule for the task has expired.</td>
</tr>
<tr>
<td></td>
<td><strong>TFC</strong> The task completed and will not run again because the task was canceled by the server.</td>
</tr>
<tr>
<td></td>
<td><strong>CR - Command Result</strong> The result_code and result_text are populated with values specific to the type of command.</td>
</tr>
<tr>
<td></td>
<td><strong>CE - Command Error</strong> The result_code and result_text are populated with values specific to the type of command.</td>
</tr>
<tr>
<td>Result</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>agent_id</td>
<td>VARCHAR(128). The ID of the agent that produced this event.</td>
</tr>
<tr>
<td>remote_id</td>
<td>VARCHAR(128). The ID of the remote database that the event applies to. This is only set for task-related events that target a specific remote database.</td>
</tr>
<tr>
<td>task_name</td>
<td>VARCHAR(128). The name of the task. This is only set for task-related events.</td>
</tr>
<tr>
<td>command_number</td>
<td>INTEGER. The command number within a task that this event applies to. This is only set for command-specific events.</td>
</tr>
<tr>
<td>run_number</td>
<td>BIGINT. The unique number assigned to each run of a task. This is only set for task-specific events.</td>
</tr>
<tr>
<td>duration</td>
<td>INTEGER. The amount of time taken by the event. This is only set for command-specific events.</td>
</tr>
<tr>
<td>event_time</td>
<td>TIMESTAMP. The time the event took place. For most events the time is based on the clock of the computer the agent is executing on.</td>
</tr>
<tr>
<td>event_received</td>
<td>TIMESTAMP. The time the event was received by the server. This is always set from the clock of the consolidated database.</td>
</tr>
<tr>
<td>result_code</td>
<td>BIGINT. An event-specific BIGINT. For example, for a SQL query command result, the code would be the SQLCODE.</td>
</tr>
<tr>
<td>result_text</td>
<td>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.</td>
</tr>
<tr>
<td>p_crsr</td>
<td>SYS_REF_CURSOR. This is an OUT parameter for Oracle only.</td>
</tr>
</tbody>
</table>

**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.

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 644
- “`ml_ra_get_task_results` system procedure” on page 644

**ml_ra_get_agent_ids system procedure**

Use this procedure to get all the agents in the consolidated database.
Returns

<table>
<thead>
<tr>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_id</td>
<td>VARCHAR(128). The agent ID.</td>
</tr>
<tr>
<td>remote_id</td>
<td>VARCHAR(128). The remote ID of the agent database.</td>
</tr>
<tr>
<td>last_download_time</td>
<td>TIMESTAMP. The last download time.</td>
</tr>
<tr>
<td>last_upload_time</td>
<td>TIMESTAMP. The last upload time.</td>
</tr>
<tr>
<td>active_task_count</td>
<td>INTEGER. The number of active tasks.</td>
</tr>
<tr>
<td>description</td>
<td>VARCHAR(2048). Reserved for future use.</td>
</tr>
<tr>
<td>p_crsr</td>
<td>SYS_REF_CURSOR. This is an OUT parameter for Oracle only.</td>
</tr>
</tbody>
</table>

See also
- “ml_ra_get_remote_ids system procedure” on page 644

ml_ra_get_agent_properties system procedure

Use this procedure to see all the properties set for an agent.

Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_id</td>
<td>VARCHAR(128). This is an IN parameter that specifies the ID of the agent you are setting properties for.</td>
</tr>
</tbody>
</table>

Returns

<table>
<thead>
<tr>
<th>Results</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>property_name</td>
<td>VARCHAR(128). The property name.</td>
</tr>
<tr>
<td>property_value</td>
<td>VARCHAR(2048). The value of the property.</td>
</tr>
<tr>
<td>last_modified</td>
<td>TIMESTAMP. The time the property was last modified.</td>
</tr>
<tr>
<td>p_crsr</td>
<td>SYS_REF_CURSOR. This is an OUT parameter for Oracle only.</td>
</tr>
</tbody>
</table>
See also

- “ml_ra_get_agent_ids system procedure” on page 641
- “ml_ra_clone_agent_properties system procedure” on page 637

### ml_ra_get_latest_event_id system procedure

Use this procedure to help determine how many new events there are.

#### Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event_id</td>
<td>BIGINT. This is an OUT parameter that specifies the ID of the latest event.</td>
</tr>
</tbody>
</table>

#### 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 639

### 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

<table>
<thead>
<tr>
<th>Results</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remote_id</td>
<td>VARCHAR(128). The remote ID.</td>
</tr>
<tr>
<td>orig_agent_id</td>
<td>VARCHAR(128). The ID of the original agent the agent database was associated with.</td>
</tr>
<tr>
<td>last_sync</td>
<td>TIMESTAMP. The time of the last synchronization.</td>
</tr>
<tr>
<td>p_crsr</td>
<td>SYS_REF_CURSOR. This is an OUT parameter for Oracle only.</td>
</tr>
</tbody>
</table>

#### 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 649

**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**

<table>
<thead>
<tr>
<th>Results</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remote_id</td>
<td>VARCHAR(128). The remote ID of the remote database.</td>
</tr>
<tr>
<td>schema_name</td>
<td>VARCHAR(128). The type of the remote database.</td>
</tr>
<tr>
<td>agent_id</td>
<td>VARCHAR(128). The agent ID of the remote database.</td>
</tr>
<tr>
<td>agent_conn_str</td>
<td>VARCHAR(2048). The agent connection string.</td>
</tr>
<tr>
<td>last_download_time</td>
<td>TIMESTAMP. The last download time.</td>
</tr>
<tr>
<td>last_upload_time</td>
<td>TIMESTAMP. The last upload time.</td>
</tr>
<tr>
<td>description</td>
<td>VARCHAR(128). The description of the database.</td>
</tr>
<tr>
<td>p_crsr</td>
<td>SYS_REF_CURSOR. This is an OUT parameter for Oracle only.</td>
</tr>
</tbody>
</table>

See also

- “ml_ra_get_agent_properties system procedure” on page 642

**ml_ra_get_task_results system procedure**

Use this procedure to get events related to a specific run of a task.

**Parameters**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_id</td>
<td>VARCHAR(128). This is an IN parameter that specifies the ID of the agent you want to get results for.</td>
</tr>
</tbody>
</table>
### Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task_name</td>
<td>VARCHAR(128). This is an IN parameter that specifies the name of the task you want to get results for.</td>
</tr>
<tr>
<td>run_number</td>
<td>INTEGER. This is an IN parameter that specifies the run number you want results for.</td>
</tr>
</tbody>
</table>

### Returns

<table>
<thead>
<tr>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>event_id</td>
<td>BIGINT. A unique ID assigned to each event. The value is incremented by 1 for each new event.</td>
</tr>
<tr>
<td>event_class</td>
<td>VARCHAR(1). The event class. The class can be either I for information or E for error.</td>
</tr>
<tr>
<td>event_type</td>
<td>VARCHAR(8). The event type.</td>
</tr>
<tr>
<td>agent_id</td>
<td>VARCHAR(128). The ID of the agent that produced this event.</td>
</tr>
<tr>
<td>remote_id</td>
<td>VARCHAR(128). The ID of the remote database that the event applies to. This is only set for task-related events that target a specific remote database.</td>
</tr>
<tr>
<td>task_name</td>
<td>VARCHAR(128). The name of the task. This is only set for task-related events.</td>
</tr>
<tr>
<td>command_number</td>
<td>INTEGER. The command number within a task that this event applies to. This is only set for command-specific events.</td>
</tr>
<tr>
<td>run_number</td>
<td>BIGINT. The unique number assigned to each run of a task. This is only set for task-specific events.</td>
</tr>
<tr>
<td>duration</td>
<td>INTEGER. The amount of time taken by the event. This is only set for command-specific events.</td>
</tr>
<tr>
<td>event_time</td>
<td>TIMESTAMP. The time the event took place. For most events the time is based on the clock of the computer the agent is executing on.</td>
</tr>
<tr>
<td>event_received</td>
<td>TIMESTAMP. The time the event was received by the server. This is always set from the clock of the consolidated database.</td>
</tr>
<tr>
<td>result_code</td>
<td>BIGINT. An event-specific BIGINT. For example, for a SQL query command result, the code would be the SQLCODE.</td>
</tr>
<tr>
<td>result_text</td>
<td>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.</td>
</tr>
</tbody>
</table>
**Result** | **Description**
---|---
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 639

**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**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_id</td>
<td>VARCHAR(128). This is an IN parameter that specifies the ID of the agent you want to get status for.</td>
</tr>
<tr>
<td>task_name</td>
<td>VARCHAR(128). This is an IN parameter that specifies the name of the task you want to get status for.</td>
</tr>
</tbody>
</table>

**Returns**

<table>
<thead>
<tr>
<th>Result</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_id</td>
<td>VARCHAR(128). The ID of the agent that produced this event.</td>
</tr>
<tr>
<td>remote_id</td>
<td>VARCHAR(128). The ID of the remote database that the event applies to.</td>
</tr>
<tr>
<td>task_name</td>
<td>VARCHAR(128). The name of the task.</td>
</tr>
<tr>
<td>task_id</td>
<td>BIGINT. The task ID.</td>
</tr>
</tbody>
</table>
### Result Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>state</td>
<td>VARCHAR(4). The state of the deployed task. State can be one of the following:</td>
</tr>
<tr>
<td></td>
<td>• P Pending. Awaiting confirmation that the agent has received the task.</td>
</tr>
<tr>
<td></td>
<td>• A Active. The agent has the task and will run it when it is scheduled to run.</td>
</tr>
<tr>
<td></td>
<td>• S Succeeded. The task is complete and will not run again unless it is reassigned.</td>
</tr>
<tr>
<td></td>
<td>• F Failed. The task is complete and will not run again unless it is reassigned.</td>
</tr>
<tr>
<td></td>
<td>• CP Cancel pending. Waiting for confirmation that the agent has canceled the task.</td>
</tr>
<tr>
<td></td>
<td>• C Canceled. The task is complete and will not run again unless it is reassigned.</td>
</tr>
<tr>
<td></td>
<td>• E Expired. The task is complete and will not run again unless it is reassigned.</td>
</tr>
<tr>
<td>reported_exec_count</td>
<td>BIGINT. The reported number of tasks that have been executed.</td>
</tr>
<tr>
<td>reported_error_count</td>
<td>BIGINT. The reported number of errors.</td>
</tr>
<tr>
<td>reported_attempt_count</td>
<td>BIGINT. The reported number of attempts to execute a task.</td>
</tr>
<tr>
<td>last_status_update</td>
<td>TIMESTAMP. The time the last status update was given.</td>
</tr>
<tr>
<td>last_success</td>
<td>TIMESTAMP. The time of the last successful task.</td>
</tr>
<tr>
<td>assignment_time</td>
<td>TIMESTAMP. The time the task was assigned.</td>
</tr>
<tr>
<td>p_crsr</td>
<td>SYS_REF_CURSOR. This is an OUT parameter for Oracle only.</td>
</tr>
</tbody>
</table>

### Remarks

The @agent_id and @task_name parameters can be set to null 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 so 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.
ml_ra_manage_remote_db system procedure

Use this procedure to add an agent-managed remote database.

Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_id</td>
<td>VARCHAR(128). This is an IN parameter that specifies the ID of the new agent to be defined in the consolidated database.</td>
</tr>
<tr>
<td>schema_name</td>
<td>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.</td>
</tr>
<tr>
<td>conn_str</td>
<td>VARCHAR(128). This is an IN parameter that specifies the database connection string used by the agent to connect to the remote database.</td>
</tr>
</tbody>
</table>

See also

- “ml_ra_add_agent_id system procedure” on page 634
- “ml_ra_clone_agent_properties system procedure” on page 637

ml_ra_notify_agent_sync system procedure

Use this procedure to cause an agent to synchronize its state.

Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_id</td>
<td>VARCHAR(128). This is an IN parameter that specifies the ID of the agent you want to synchronize.</td>
</tr>
</tbody>
</table>

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 644
ml_ra_notify_task system procedure

Use this procedure to run a task using server initiated remote tasks (SIRT).

Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_id</td>
<td>VARCHAR(128). This is an IN parameter that specifies the ID of the agent you want to run the task.</td>
</tr>
<tr>
<td>task_name</td>
<td>VARCHAR(128). This is an IN parameter that specifies the name of the task you want the agent to execute.</td>
</tr>
</tbody>
</table>

See also
- “ml_ra_cancel_notification system procedure” on page 636
- “ml_ra_delete_task system procedure” on page 639

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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>taskdb_remote_id</td>
<td>VARCHAR(128). This is an IN parameter that specifies the remote ID of the orphaned agent database.</td>
</tr>
<tr>
<td>new_agent_id</td>
<td>VARCHAR(128). This is an IN parameter that specifies the ID of the new agent you want to assign the orphaned agent database to.</td>
</tr>
</tbody>
</table>

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 643

ml_ra_set_agent_property system procedure

Use this procedure to set remote agent properties.
Parameters

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>agent_id</td>
<td>VARCHAR(128). This is an IN parameter that specifies the agent ID.</td>
</tr>
<tr>
<td>property_name</td>
<td>VARCHAR(128). This is an IN parameter that specifies the property name to be set.</td>
</tr>
<tr>
<td>property_value</td>
<td>VARCHAR(2048). This is an IN parameter that specifies the property value to be set.</td>
</tr>
</tbody>
</table>

Remarks

The agent supports the following properties:

- **mlstream**  The MobiLink stream parameters, for example \texttt{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 637

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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>remote_id</td>
<td>VARCHAR(128). This is an IN parameter that specifies the remote ID to be severed.</td>
</tr>
<tr>
<td>schema_name</td>
<td>VARCHAR(128). This is an IN parameter that indicates the type of the remote database.</td>
</tr>
</tbody>
</table>

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 648

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

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>user</td>
<td>VARCHAR(128). The MobiLink user name.</td>
</tr>
<tr>
<td>remote_id</td>
<td>VARCHAR(128). The remote ID.</td>
</tr>
</tbody>
</table>

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

MobiLink server includes the following utilities:

- “MobiLink Stop utility (mlstop)”
- “MobiLink User Authentication utility (mluser)”
- “MobiLink Replay utility (mlreplay)”
- “MobiLink Generated Replay API utility (mlgenreplayapi)”
- “MobiLink Arbiter Server utility for Windows (mlarbiter)”
- “MobiLink Arbiter Server utility for Unix (mlarbiter.sh)”
- “MobiLink Arbiter Stop utility (mlarbstop)” on page 665

See also

- “MobiLink client utilities” [MobiLink - Client Administration]
- “UltraLite utilities” [UltraLite - Database Management and Reference]
- “Certificate utilities” [SQL Anywhere Server - Database Administration]
- “Database administration utilities” [SQL Anywhere Server - Database Administration]

MobiLink Stop utility (mlstop)

Stops the MobiLink server on the local computer.

Syntax

\texttt{mlstop [ \texttt{options} ] [ \texttt{name} ]}

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@data</td>
<td>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 “Configuration files” [SQL Anywhere Server - Database Administration]. 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].</td>
</tr>
<tr>
<td>-h</td>
<td>Hard shutdown. MobiLink stops all synchronizations and exits. Some remotes may report an error.</td>
</tr>
<tr>
<td>-q</td>
<td>Quiet mode. This suppresses the banner.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-------------</td>
</tr>
<tr>
<td><code>-t time</code></td>
<td>Soft shutdown, with a hard shutdown after the specified time. <code>time</code> is a number followed by D, H, M, or S (for days, hours, minutes and seconds). For example, <code>-t 10m</code> 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.</td>
</tr>
<tr>
<td><code>-w</code></td>
<td>Waits for the MobiLink server to shut down before returning from the command.</td>
</tr>
<tr>
<td><code>name</code></td>
<td>If the MobiLink server is started using the <code>-zs</code> option, it must be shut down by specifying the same server name. See “-zs mlsrv16 option” on page 95.</td>
</tr>
</tbody>
</table>

Remarks

By default (if neither `-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 ] }
```
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@data</td>
<td>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 “Configuration files” [SQL Anywhere Server - Database Administration]. 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].</td>
</tr>
<tr>
<td>-c &quot;keyword=value;...&quot;</td>
<td>Use this to supply database connection parameters. The connection string must provide the utility sufficient privileges to connect to the consolidated database using an ODBC data source. This parameter is required.</td>
</tr>
<tr>
<td>-d</td>
<td>Deletes the user name(s) specified by -f or -u. This option cannot be used with mluser -r option.</td>
</tr>
<tr>
<td>-f filename</td>
<td>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.</td>
</tr>
<tr>
<td>-fips</td>
<td>When set, mluser fails if support for FIPS-certified encryption is not installed.</td>
</tr>
<tr>
<td>-n user authentication policy name</td>
<td>Registers a MobiLink user with LDAP user authentication.</td>
</tr>
<tr>
<td>-o filename</td>
<td>Logs output messages to the specified file.</td>
</tr>
<tr>
<td>-ot filename</td>
<td>Truncate the log file and then append output messages to it. The default is to send output to the screen.</td>
</tr>
<tr>
<td>-pc collation-id</td>
<td>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 “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.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>-p password</code></td>
<td>Password to associate with the user. This option can only be used with <code>-u</code>.</td>
</tr>
<tr>
<td><code>-r remote-id</code></td>
<td>Use this option with <code>-u username</code> and mluser resets the synchronization state for the given remote ID and username. The last_upload_time and last_download_time columns in the ml_subscription table are reset to their default values for the given username and remote ID. The default values for the progress, last_upload_time, and last_download_time columns are 0, '1900/01/01 00:00:00', and '1900/01/01 00:00:00', respectively. This option cannot be used with the mluser <code>-d</code> option.</td>
</tr>
<tr>
<td><code>-u username</code></td>
<td>Specify the user name to add (or delete, if used with <code>-d</code>). Only one user can be specified on a single command line. This option is used with <code>-p</code> if passwords are being used. You must specify either <code>-f</code> or <code>-u</code>.</td>
</tr>
<tr>
<td><code>-v</code></td>
<td>Specifies verbose logging.</td>
</tr>
</tbody>
</table>

**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 mlsrv16. 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 mlsrv16 option” on page 95
- “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.

**Syntax**
```mlreplay [options] [name=value [name2=value2...]] [[dll_name] filename]```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>@data</code></td>
<td>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 “Configuration files” [SQL Anywhere Server - Database Administration]. 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].</td>
</tr>
<tr>
<td><code>-ap</code></td>
<td>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 and adjust sequence numbers to avoid sequence number errors.</td>
</tr>
<tr>
<td><code>-f time_scale_factor</code></td>
<td>A multiplier evenly applied to recorded times.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>-ldt</code> <code>last_download_time</code></td>
<td>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 <code>mlreplay</code> during the replay session. Even if the <code>-ldt</code> option is not used, <code>mlreplay</code> 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 <code>-sci</code> option is used) or by the IdentifySimulatedClient callback when a DLL is provided.</td>
</tr>
<tr>
<td><code>-ls</code></td>
<td>Log the total running time, total time spent replaying, the total number of repetitions that either completed successfully, failed, or skipped for each simulated client. <code>mlreplay</code> still logs this information before exiting even when this option is not specified.</td>
</tr>
<tr>
<td><code>-n number_of_simulated_clients</code></td>
<td>The number of simulated clients to run. The minimum is 1.</td>
</tr>
<tr>
<td></td>
<td>This option can be used with the <code>-sci</code> option when the number of simulated clients specified by <code>-n</code> less than or equal to the number of simulated clients in the simulated client information file. When used together, <code>-n</code> specifies the number of simulated clients run. These options allow one simulated client information file, specifying <code>x</code> number of simulated clients, to replay a protocol with <code>1</code> to <code>x</code> simulated clients.</td>
</tr>
<tr>
<td><code>-o file</code></td>
<td>Log command line options and output messages to the specified file.</td>
</tr>
<tr>
<td><code>-os size</code></td>
<td>Limit the maximum size of a log file. When the log reaches the specified size (minimum 10 KB), it is renamed to <code>YYMDDxx.rlg</code> and a new log file is started with the original name.</td>
</tr>
<tr>
<td><code>-ot file</code></td>
<td>Truncate the log file. Log command line options and output messages to the specified file.</td>
</tr>
<tr>
<td><code>-p password</code></td>
<td>Replace passwords with the given password.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| -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.                                                                 |
| -rep number_of_repetitions | Specify the number of times simulated clients should replay the recorded protocol. Each repetition can be customized if a replay DLL/shared object is used.  
When using the generated replay API, the GetUploadTransaction, GetDownloadApplyTime, ReportEndOfReplay, and DelayStartOfReplay callbacks are called for each repetition. |
| -rg                | Replace remote IDs with a GUID.                                                                                                              |
| -rnt seconds       | Instruct simulated clients to start new repetitions of protocol replays until the given number of seconds is reached. Simulated clients are not stopped but no additional repetitions are started.  
When specified, the numRepetitions parameter of any API callbacks is set to 0 |
| -rp pattern        | Replace the given pattern in usernames, passwords, and remote IDs specified on the command line with the simulated client number.  |

MobiLink reference

658 Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-sci file</td>
<td>Provide mlreplay with a list of user names, passwords, remote IDs, last download times, and script versions to use for replaying. mlreplay creates a simulated client for each line in the file to replay the recorded protocol with that client information. The format of each line should be: [username],[password],[remote ID],[last download time],[script version]. The format of the last download time should be yyyy-MM-dd hh:mm:ss.SSS. If the username, password, last download time, or script version fields are left blank, mlreplay uses the corresponding values in the recorded protocol. If the remote ID is left blank, mlreplay replaces the remote ID with a GUID. The -u, -p, -r, -rg, -ldt and -sv options cannot be used with this option nor can a DLL. This option can be used with the -n option when the number of simulated clients specified by -n is less than or equal to the number of simulated clients in the simulated client information file. When used together, -n specifies the number of simulated clients run. These options allow one simulated client information file, specifying x number of simulated clients, to replay a protocol with 1 to x simulated clients.</td>
</tr>
<tr>
<td>-sv script version</td>
<td>Replace script versions with the given script version.</td>
</tr>
<tr>
<td>-u user name</td>
<td>Replace user names with the given user name.</td>
</tr>
<tr>
<td>-x stream(opts)</td>
<td>The protocol stream and stream options to use to connect to the MobiLink server. The liveness timeout can be set with this option and is automatically adjusted based on what the MobiLink server is using.</td>
</tr>
</tbody>
</table>

**Remarks**

The optional dll_name parameter is the name of the replay DLL you want mlreplay to use. The replay DLL is compiled from the Replay API. See “MobiLink Generated Replay API utility (mlgenreplayapi)” on page 662.

The name=value pairs are like command line arguments for the replay API. They are accessible in all mlreplay callbacks and can be used to customize the behavior of the replay DLL. They are only used if a replay DLL is used. For example, to use the same replay DLL to perform synchronizations to different databases (with different instances of mlreplay) and at the end of the synchronizations you want to connect to the database to make sure the data was uploaded successfully, you could use a name=value pair to specify the connection string for the database rather than hard coding it in the replay DLL.
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 mlsrv16 option” on page 70.

In addition to the data sent to and from the MobiLink server, the recorded protocol file also contains timing information so that 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 different options. The simulated client information consists of the username, password, remote ID, last download time, and script version. This information can be customized using the `-u`, `-p`, `-r` (or `-rg`), `-ldt`, and `-sv` options respectively.

The mlreplay utility can replay a recorded protocol file concurrently using multiple different simulated clients. There are three ways to do this:

- **Using only the command line** You can concurrently replay a recorded protocol file by using a combination of the options `-n`, `-u`, `-p`, `-sv`, `-r`, `-rg`, and `-rp`. The `-n` option is used to specify the number of simulated clients, whereas `-u`, `-p`, `-sv`, `-r` and `-rg` are used to specify information about each client. By default, you can specify an asterisk (*) when using `-u`, `-p`, `-sv`, and/or `-r` (as many times as you want), to tell mlreplay to replace the asterisk with the simulated client number. You can change the asterisk to any other character using the `-rp` option).

For example, `mlreplay -ap -x tcpip -n 2 -rp $ -u user_$ -p pwd_$ -r rid_ $ -sv test_script recorded_protocol.mlr` runs mlreplay with two simulated clients. Simulated client 1 has the following information:

- user: user_1
- password: pwd_1
- remote id: rid_1
- script version: test_script

Simulated client 2 has the following information:

- user: user_2
- password: pwd_2
- remote id: rid_2
- script version: test_script

The following rules are used if any of the options are omitted:

- When a username, password, or script version is not specified, simulated clients use the username, password, or script version that is recorded in the recorded protocol file.
- When a remote ID is not specified and the number of simulated clients is greater than 1, a different GUID is automatically generated for each remote ID. When the number of simulated clients is 1,
the remote ID recorded in the recorded protocol file is used. You can force a GUID to be generated by using the -rg option.

- When a username, password, remote ID, or script version is specified but does not contain an asterisk (*) (or whatever character was specified by the -rp option), each simulated client uses the same username, password, remote ID or script version.

**Using a simulated client information file** You can concurrently replay a recorded protocol file by specifying a simulated client information file by using the -sci option. The simulated client information file is a .csv file where each line has a username, password, remote ID, last download time, and script version (in that order).

The mlreplay utility fills in any blank fields with the same rules described under the **Using only the command line** option.

By default, mlreplay creates a simulated client for each line of information in the simulated client information file. However, you can use the -n option along with the -sci file to limit the number of simulated clients. If your simulated client information file specifies x simulated clients, you can use the -n option to specify a number from 1 to x, so that mlreplay only uses that number of simulated clients.

Using a simulated client information file is more flexible than using just the command line but less flexible than using a replay DLL.

**Using a replay DLL** When using a replay DLL, you use the -n option to specify the number of simulated clients. All other information is retrieved when mlreplay calls into the user implemented callbacks. This approach provides the greatest flexibility as it allows other parts of the replay to be customized. See “MobiLink Replay C++ callbacks” on page 597 and “MobiLink Generated Replay API utility (mlgenreplayapi)” on page 662.

The mlreplay utility can run multiple simulated clients to replay a protocol at the command line. The number of simulated clients to run can be specified by the -n option. Use the asterisk character to denote the simulated client number when specifying usernames, passwords, remote IDs, and script versions with the -u, -p, -r, and -sv options, respectively. The username, password, remote ID, and script version for each simulated client are determined by the following rules:

- When a username or password is not specified, all simulated clients use the username or password recorded in the recorded protocol file being replayed.

- When a remote ID is not specified and the number of simulated clients is greater than 1, each remote ID becomes an automatically generated GUID. When the number of simulated clients is 1, the remote ID recorded in the recorded protocol file is used; you can force a GUID value with the -rg option.

- When the specified username, password, or remote ID does not contain an asterisk, simulated clients use the same username, password, or remote ID. When the specified username, password, or remote ID contains at least one asterisk, simulated clients get their own unique username, password, or remote ID where each asterisk is replaced with the simulated client number.

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.
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.

Further customizations can be made to the replay session using the MobiLink Generated Replay API utility.

**See also**

- “MobiLink Generated Replay API utility (mlgenreplayapi)” on page 662
- “-rrp mlsrv16 option” on page 69
- “-rp mlsrv16 option” on page 70
- “-lsc mlsrv16 option” on page 59

**MobiLink Generated Replay API utility (mlgenreplayapi)**

The mlgenreplayapi tool reads a recorded protocol file and generates the MobiLink Replay API for the schema in that file.

**Syntax**

`mlgenreplayapi [options] filename`

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@data</td>
<td>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 “Configuration files” [SQL Anywhere Server - Database Administration]. 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].</td>
</tr>
<tr>
<td>-d directory</td>
<td>The directory to output the generated files.</td>
</tr>
<tr>
<td>-o file</td>
<td>Logs command line options and output messages to the specified file.</td>
</tr>
</tbody>
</table>
**Option** | **Description**
--- | ---
-ossize | Limits the maximum size of a log file. When the log reaches the specified size (minimum 10 KB), it is renamed to YYMMDDxx.rlg and a new log file is started with the original name.

-ot file | Truncates the log file. Appends command line options and output messages to the specified file. The default is to send output to the screen.

**Remarks**

The API can be modified (only the code in mtreplaycallbacks.cpp needs to be modified) to customize the data uploaded to the MobiLink server during the replay session. The Replay API can then be 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 also**

- “MobiLink Replay C++ callbacks” on page 597
- “MobiLink Replay utility (mlreplay)” on page 656

**MobiLink Arbiter Server utility for Windows (mlarbiter)**

The mlarbiter command starts the MobiLink arbiter server.

**Syntax**

mlarbiter

**Remarks**

The MobiLink arbiter listens on port 4953 by default.

The MobiLink arbiter server ensures that only a single MobiLink server in a server farm is running as the primary server, preventing redundant notifications in a server-initiated synchronization environment.

This command is used with the MobiLink server -ca option, which provides the MobiLink server with the host name of the arbiter.

If the MobiLink server is not able make a connection to the arbiter after the arbiter starts, the MobiLink server 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 elect a primary server, the primary server shuts down immediately and the secondary servers try to re-establish the arbiter connection every 15 seconds. After a connection to the arbiter is established, the MobiLink servers re-elect a primary server.
Example

The following example shows how to use the MobiLink arbiter server with a MobiLink server farm.

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.
   
   `mlsrv16 -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 mlsrv16 option” on page 48
- “-lsc mlsrv16 option” on page 59
- “Architecture” [MobiLink - Getting Started]

MobiLink Arbiter Server utility for Unix (`mlarbiter.sh`)

The `mlarbiter.sh` command starts and stops the MobiLink arbiter server.

Syntax

```
mlarbiter.sh [ option ]
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>Starts the MobiLink Arbiter Server utility.</td>
</tr>
<tr>
<td>stop</td>
<td>Stops the MobiLink Arbiter Server utility.</td>
</tr>
</tbody>
</table>

Remarks

The MobiLink arbiter listens on port 4953 by default.

This command is used with the `mlsrv16 -ca` option, which provides the MobiLink server with the host name of the arbiter.

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.

If the MobiLink server is not able make a connection to the arbiter after the arbiter starts, the MobiLink server 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 try to re-establish the arbiter.
connection every 15 seconds. After a connection to the arbiter is established, the MobiLink servers re-elect a primary server.

Example
The following example shows how to use the MobiLink arbiter server with a MobiLink server farm.

1. Start the MobiLink arbiter on a computer with the following command line.
   
   ```
   mlarbiter.sh start
   ```

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.
   
   ```
   mlsrv16 -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 mlsrv16 option” on page 48
- “-lsc mlsrv16 option” on page 59
- “Architecture” [MobiLink - Getting Started]

**MobiLink Arbiter Stop utility (mlarbstop)**

The mlarbstop command is used to stop the MobiLink arbiter server.

**Syntax**

```
mlarbstop [ option ]
```

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-y</code></td>
<td>Stops the MobiLink arbiter server immediately, even if there are connections to it.</td>
</tr>
</tbody>
</table>

**Remarks**

The mlarbstop utility can be used to stop the MobiLink arbiter server when it is running on the local computer.

If you run mlarbstop without the `-y` option and there are no connections to the arbiter server, the arbiter server is stopped immediately.

If you run mlarbstop without the `-y` option and there are connections to the arbiter server, MobiLink server issues an error.

**Example**

The following example shows how to stop the MobiLink arbiter server on a local computer.
1. Stop the MobiLink arbiter on a local computer with the following command line.

   `mlarbstop -y`

See also
- “MobiLink Arbiter Server utility for Windows (mlarbiter)” on page 663
- “MobiLink Arbiter Server utility for Unix (mlarbiter.sh)” on page 664
- “-ca mlsrv16 option” on page 48
- “-lsc mlsrv16 option” on page 59
- “Architecture” [MobiLink - Getting Started]

**MobiLink data mappings between remote and consolidated databases**

Depending on the consolidated database you are using, the MobiLink server may map a specified data type to a different data type. The following sections show the data type mappings between SQL Anywhere and/or UltraLite and the supported consolidated databases.

- “Adaptive Server Enterprise data mapping”
- “IBM DB2 LUW data mapping”
- “Microsoft SQL Server data mapping”
- “MySQL data mapping”
- “Oracle data mapping”
- “SAP Sybase IQ Enterprise data mapping” on page 708

### 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 LONG VARCHAR on the remote database should be type TEXT 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.

<table>
<thead>
<tr>
<th>SQL Anywhere or UltraLite data type</th>
<th>Adaptive Server Enterprise data type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>BIGINT</td>
<td></td>
</tr>
<tr>
<td>SQL Anywhere or UltraLite data type</td>
<td>Adaptive Server Enterprise data type</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>BIT</td>
<td>BIT</td>
<td></td>
</tr>
<tr>
<td>BINARY((n&lt;\text{MCL}))</td>
<td>BINARY((n))</td>
<td></td>
</tr>
<tr>
<td>BINARY((n&gt;\text{MCL}))</td>
<td>IMAGE</td>
<td></td>
</tr>
<tr>
<td>CHAR((n&lt;\text{MCL}))</td>
<td>VARCHAR((n))</td>
<td></td>
</tr>
<tr>
<td>CHAR((n&gt;\text{MCL}))</td>
<td>TEXT</td>
<td>On download, ensure the values are not too long.</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
<td>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.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>DATETIME(^1) or BIG-DATETIME(^2)</td>
<td>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.</td>
</tr>
<tr>
<td>DECIMAL((p&lt;39, s))</td>
<td>DECIMAL((p, s))</td>
<td>The precision of the Adaptive Server Enterprise NUMERIC can be from 1 to 38 digits ((p&lt;39)).</td>
</tr>
<tr>
<td>DECIMAL((p\geq39, s))</td>
<td></td>
<td>There is no corresponding data type in Adaptive Server Enterprise.</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE PRECISION</td>
<td></td>
</tr>
<tr>
<td>SQL Anywhere or UltraLite data type</td>
<td>Adaptive Server Enterprise data type</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>FLOAT($p$)</td>
<td>FLOAT($p$)</td>
<td></td>
</tr>
<tr>
<td>IMAGE</td>
<td>IMAGE</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>LONG BINARY</td>
<td>IMAGE</td>
<td></td>
</tr>
<tr>
<td>LONG NVARCHAR</td>
<td>UNITEXT</td>
<td></td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td>TEXT</td>
<td></td>
</tr>
<tr>
<td>MONEY</td>
<td>MONEY</td>
<td></td>
</tr>
<tr>
<td>NCHAR($c$&lt;=$MCL)</td>
<td>UNIVARCHAR($c/2$)</td>
<td></td>
</tr>
<tr>
<td>NCHAR($c$&gt;$MCL)</td>
<td>UNITEXT</td>
<td>On download, ensure the values are not too long.</td>
</tr>
<tr>
<td>NTEXT</td>
<td>UNITEXT</td>
<td></td>
</tr>
<tr>
<td>NUMERIC($p$&lt;$39$, $s$)</td>
<td>NUMERIC($p$, $s$)</td>
<td>The precision of the Adaptive Server Enterprise decimal can be from 1 to 38 digits ($p$$&lt;$39).</td>
</tr>
<tr>
<td>NUMERIC($p$&lt;=$39$, $s$)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NVARCHAR($c$&lt;=$MCL)</td>
<td>UNIVARCHAR($c/2$)</td>
<td></td>
</tr>
<tr>
<td>NVARCHAR($c$&gt;$MCL)</td>
<td>UNIVARCHAR($c/2$)</td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>SQL Anywhere or UltraLite data type</td>
<td>Adaptive Server Enterprise data type</td>
<td>Notes</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>SMALLDATETIME</td>
<td>DATETIME(^1) or BIGDATETIME(^2)</td>
<td>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.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SMALLINT</td>
<td></td>
</tr>
<tr>
<td>SMALLMONEY</td>
<td>SMALLMONEY</td>
<td></td>
</tr>
<tr>
<td>TEXT</td>
<td>TEXT</td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>TIME(^1) or BIGTIME(^2)</td>
<td>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.</td>
</tr>
<tr>
<td>SQL Anywhere or UltraLite data type</td>
<td>Adaptive Server Enterprise data type</td>
<td>Notes</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>--------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>DATETIME(^1) or BIG-DATETIME(^2)</td>
<td>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.</td>
</tr>
<tr>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>VARCHAR(34)</td>
<td>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.</td>
</tr>
<tr>
<td>TINYINT</td>
<td>TINYINT</td>
<td></td>
</tr>
<tr>
<td>UNIQUEIDENTIFIER</td>
<td>CHAR(36)</td>
<td></td>
</tr>
<tr>
<td>UNIQUEIDENTIFIERSTR</td>
<td>CHAR(36)</td>
<td>Do not use UNIQUEIDENTIFIERSTR. Use UNIQUEIDENTIFIER instead.</td>
</tr>
<tr>
<td>UNSIGNED BIGINT</td>
<td>UNSIGNED BIGINT</td>
<td></td>
</tr>
<tr>
<td>UNSIGNED INTEGER</td>
<td>UNSIGNED INT</td>
<td></td>
</tr>
<tr>
<td>UNSIGNED SMALLINT</td>
<td>UNSIGNED SMALLINT</td>
<td></td>
</tr>
</tbody>
</table>
### 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.

<table>
<thead>
<tr>
<th>Adaptive Server Enterprise data type</th>
<th>SQL Anywhere or UltraLite data type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>BIGINT</td>
<td></td>
</tr>
<tr>
<td>BIGDATETIME(^1)</td>
<td>TIMESTAMP</td>
<td></td>
</tr>
<tr>
<td>BIGTIME(^1)</td>
<td>TIME</td>
<td></td>
</tr>
<tr>
<td>BINARY((n))</td>
<td>BINARY((n))</td>
<td></td>
</tr>
<tr>
<td>BIT</td>
<td>BIT</td>
<td></td>
</tr>
</tbody>
</table>

1 Only applies to Adaptive Server Enterprise before version 15.5.

2 Only applies to Adaptive Server Enterprise version 15.5 and later.
<table>
<thead>
<tr>
<th>Adaptive Server Enterprise data type</th>
<th>SQL Anywhere or UltraLite data type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR($n$)</td>
<td>VARCHAR($n$)</td>
<td>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.</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
<td>For SQL Anywhere and UltraLite, the time value must in the format 00:00:00.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>DATETIME</td>
<td>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.</td>
</tr>
<tr>
<td>DECIMAL($p$,$s$)</td>
<td>DECIMAL($p$,$s$)</td>
<td></td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>DOUBLE</td>
<td></td>
</tr>
<tr>
<td>FLOAT($p$)</td>
<td>FLOAT($p$)</td>
<td></td>
</tr>
<tr>
<td>IMAGE</td>
<td>LONG BINARY</td>
<td></td>
</tr>
<tr>
<td>INT</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>MONEY</td>
<td>MONEY</td>
<td></td>
</tr>
<tr>
<td>Adaptive Server Enterprise data type</td>
<td>SQL Anywhere or UltraLite data type</td>
<td>Notes</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>NCHAR(n)</td>
<td>VARCHAR(n)</td>
<td>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.</td>
</tr>
<tr>
<td>NUMERIC(p,s)</td>
<td>NUMERIC(p,s)</td>
<td></td>
</tr>
<tr>
<td>NVARCHAR(n)</td>
<td>VARCHAR(n)</td>
<td>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.</td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>SMALLDATETIME</td>
<td>SMALLDATETIME</td>
<td>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.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SMALLINT</td>
<td></td>
</tr>
<tr>
<td>SMALLMONEY</td>
<td>SMALLMONEY</td>
<td></td>
</tr>
<tr>
<td>TEXT</td>
<td>LONG VARCHAR</td>
<td></td>
</tr>
<tr>
<td>Adaptive Server Enterprise data type</td>
<td>SQL Anywhere or UltraLite data type</td>
<td>Notes</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>TIME</td>
<td>TIME</td>
<td>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.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>VARBINARY(8)</td>
<td>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 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.</td>
</tr>
<tr>
<td>TINYINT</td>
<td>TINYINT</td>
<td></td>
</tr>
<tr>
<td>UNSIGNED BIGINT</td>
<td>UNSIGNED BIGINT</td>
<td></td>
</tr>
<tr>
<td>UNSIGNED INT</td>
<td>UNSIGNED INT</td>
<td></td>
</tr>
<tr>
<td>UNSIGNED SMALLINT</td>
<td>UNSIGNED SMALLINT</td>
<td></td>
</tr>
<tr>
<td>VARBINARY(n)</td>
<td>VARBINARY(n)</td>
<td></td>
</tr>
<tr>
<td>VARCHAR(n)</td>
<td>VARCHAR(n)</td>
<td></td>
</tr>
<tr>
<td>UNICHAR(n)</td>
<td>NVARCHAR(n)</td>
<td>Not available in UltraLite.</td>
</tr>
</tbody>
</table>

Not available in UltraLite.
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.

<table>
<thead>
<tr>
<th>SQL Anywhere or UltraLite data type</th>
<th>IBM DB2 LUW data type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>BIGINT</td>
<td></td>
</tr>
<tr>
<td>BINARY(n&lt;MRL)</td>
<td>VARCHAR(n) FOR BIT DATA</td>
<td></td>
</tr>
<tr>
<td>BINARY(n&gt;=MRL)</td>
<td>BLOB(n)</td>
<td></td>
</tr>
<tr>
<td>BIT</td>
<td>SMALLINT</td>
<td></td>
</tr>
<tr>
<td>CHAR(n&lt;MRL)</td>
<td>VARCHAR(n)</td>
<td></td>
</tr>
<tr>
<td>CHAR(n&gt;=MRL)</td>
<td>CLOB(n)</td>
<td>IBM DB2 LUW values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
<td>For SQL Anywhere and UltraLite, the time value must in the format 00:00:00.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>TIMESTAMP</td>
<td></td>
</tr>
<tr>
<td>SQL Anywhere or UltraLite data type</td>
<td>IBM DB2 LUW data type</td>
<td>Notes</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>-----------------------</td>
<td>-------</td>
</tr>
<tr>
<td>DECIMAL((p&lt;32,s))</td>
<td>DECIMAL((p,s))</td>
<td>The precision of SQL Anywhere DECIMAL is between 1 and 127. The maximum precision of IBM DB2 LUW DECIMAL is 31.</td>
</tr>
<tr>
<td>DECIMAL((p&gt;=32,s))</td>
<td></td>
<td>Any data of SQL Anywhere DECIMAL precision greater than 31 cannot be synchronized to IBM DB2 LUW.</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>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.</td>
</tr>
<tr>
<td>FLOAT(1-24)</td>
<td>REAL</td>
<td>FLOAT can cause problems if the consolidated and remote databases don’t allow the exact same (imprecise) values. Not all possible values are tested, so care must be taken. To avoid problems, do not use these types as part of a primary key.</td>
</tr>
<tr>
<td>FLOAT(25-53)</td>
<td>DOUBLE</td>
<td>FLOAT can cause problems if the consolidated and remote databases don’t allow the exact same (imprecise) values. Not all possible values are tested, so care must be taken. To avoid problems, do not use these types as part of a primary key.</td>
</tr>
<tr>
<td>IMAGE</td>
<td>BLOB((n))</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>LONG BINARY</td>
<td>BLOB((n))</td>
<td></td>
</tr>
<tr>
<td>SQL Anywhere or UltraLite data type</td>
<td>IBM DB2 LUW data type</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------</td>
<td>-------</td>
</tr>
<tr>
<td>LONG NVARCHAR</td>
<td>CLOB(n)</td>
<td>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.</td>
</tr>
<tr>
<td>LONG VARBIT</td>
<td>CLOB(n)</td>
<td></td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td>CLOB(n)</td>
<td></td>
</tr>
<tr>
<td>MONEY</td>
<td>DECIMAL(19,4)</td>
<td></td>
</tr>
<tr>
<td>NCHAR(c)</td>
<td>VARCHAR(n) or CLOB(n)</td>
<td>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 cannot 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).</td>
</tr>
<tr>
<td>NUMERIC(p&lt;32,s)</td>
<td>NUMERIC(p,s)</td>
<td></td>
</tr>
<tr>
<td>NUMERIC(p&gt;=32,s)</td>
<td></td>
<td>There is no corresponding data type in IBM DB2 LUW.</td>
</tr>
<tr>
<td>NTEXT</td>
<td>CLOB(n)</td>
<td>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.</td>
</tr>
<tr>
<td>SQL Anywhere or UltraLite data type</td>
<td>IBM DB2 LUW data type</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------</td>
<td>-------</td>
</tr>
<tr>
<td>NVARCHAR((c))</td>
<td>VARCHAR((n)) or CLOB((n))</td>
<td>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 cannot 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)).</td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>REAL can cause problems if the consolidated and remote databases don't allow the exact same (imprecise) values. Not all possible values are tested, so care must be taken. To avoid problems, do not use these types as part of a primary key.</td>
</tr>
<tr>
<td>SMALLDATETIME</td>
<td>TIMESTAMP</td>
<td></td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SMALLINT</td>
<td></td>
</tr>
<tr>
<td>SMALLMONEY</td>
<td>DECIMAL((10,4))</td>
<td></td>
</tr>
<tr>
<td>ST_GEOMETRY</td>
<td>ST_GEOMETRY</td>
<td></td>
</tr>
<tr>
<td>TEXT</td>
<td>CLOB((n))</td>
<td></td>
</tr>
<tr>
<td>SQL Anywhere or UltraLite data type</td>
<td>IBM DB2 LUW data type</td>
<td>Notes</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>----------------------</td>
<td>-------</td>
</tr>
<tr>
<td>TIME</td>
<td>TIMESTAMP or TIME</td>
<td>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. To preserve the precision of a time column, the MobiLink server always binds the TIME column with the ODBC SQLTYPE_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.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>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.</td>
</tr>
<tr>
<td>TINYINT</td>
<td>SMALLINT</td>
<td>For download, IBM DB2 LUW values must be non-negative.</td>
</tr>
<tr>
<td>UNIQUEIDENTIFIER</td>
<td>CHAR(36)</td>
<td></td>
</tr>
<tr>
<td>UNIQUEIDENTIFIERSTR</td>
<td>CHAR(36)</td>
<td>UNIQUEIDENTIFIERSTR is not recommended for IBM DB2 LUW. Use UNIQUEIDENTIFIER instead.</td>
</tr>
<tr>
<td>UNSIGNED BIGINT</td>
<td>DECIMAL(20)</td>
<td>For download, IBM DB2 LUW values must be non-negative.</td>
</tr>
<tr>
<td>UNSIGNED INTEGER</td>
<td>DECIMAL(11)</td>
<td>For download, IBM DB2 LUW values must be non-negative.</td>
</tr>
<tr>
<td>SQL Anywhere or UltraLite data type</td>
<td>IBM DB2 LUW data type</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>--------------------------------------</td>
<td>----------------------------------------------------------------------</td>
</tr>
<tr>
<td>UNSIGNED SMALLINT</td>
<td>DECIMAL(5)</td>
<td>For download, IBM DB2 LUW values must be non-negative.</td>
</tr>
<tr>
<td>UNSIGNED TINYINT</td>
<td>SMALLINT</td>
<td>For download, IBM DB2 LUW values must be non-negative.</td>
</tr>
<tr>
<td>VARBINARY($n&lt;MRL)$</td>
<td>VARCHAR($n$) FOR BIT DATA</td>
<td></td>
</tr>
<tr>
<td>VARBINARY($n&gt;=MRL)$</td>
<td>BLOB($n$)</td>
<td></td>
</tr>
<tr>
<td>VARBIT($n&lt;MRL)$</td>
<td>VARCHAR($n$)</td>
<td></td>
</tr>
<tr>
<td>VARBIT($n&gt;=MRL)$</td>
<td>CLOB($n$)</td>
<td></td>
</tr>
<tr>
<td>VARCHAR($n&lt;MRL)$</td>
<td>VARCHAR($n$)</td>
<td></td>
</tr>
<tr>
<td>VARCHAR($n&gt;=MRL)$</td>
<td>CLOB($n$)</td>
<td>IBM DB2 LUW values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.</td>
</tr>
<tr>
<td>XML</td>
<td>CLOB($n$)</td>
<td></td>
</tr>
</tbody>
</table>

**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 LUW table, you need to pay attention to the page size. IBM DB2 LUW 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 LUW documentation.

<table>
<thead>
<tr>
<th>IBM DB2 LUW data type</th>
<th>SQL Anywhere or UltraLite data type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BLOB</td>
<td>LONG BINARY</td>
<td></td>
</tr>
<tr>
<td>BIGINT</td>
<td>BIGINT</td>
<td></td>
</tr>
<tr>
<td>IBM DB2 LUW data type</td>
<td>SQL Anywhere or UltraLite data type</td>
<td>Notes</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CHAR(n)</td>
<td>VARCHAR(n)</td>
<td>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.</td>
</tr>
<tr>
<td>CHAR(n) FOR BIT DATA</td>
<td>BINARY(n)</td>
<td></td>
</tr>
<tr>
<td>CLOB(n)</td>
<td>LONG VARCHAR</td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
<td>For SQL Anywhere and UltraLite, the time value must in the format 00:00:00.</td>
</tr>
<tr>
<td>DB2GSE.ST_GEOMETRY</td>
<td>ST_GEOMETRY</td>
<td></td>
</tr>
<tr>
<td>DBCLOB(n)</td>
<td>LONG VARCHAR</td>
<td>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.</td>
</tr>
<tr>
<td>DECIMAL(p,s)</td>
<td>DECIMAL(p,s)</td>
<td></td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>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.</td>
</tr>
<tr>
<td>FLOAT</td>
<td>DOUBLE</td>
<td>FLOAT can cause problems if the consolidated and remote databases don't allow the exact same (imprecise) values. Not all possible values are tested, so care must be taken. To avoid problems, do not use these types as part of a primary key.</td>
</tr>
<tr>
<td>DB2GSE.ST_GEOMETRY</td>
<td>ST_GEOMETRY</td>
<td></td>
</tr>
<tr>
<td>DBCLOB(n)</td>
<td>LONG VARCHAR</td>
<td>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.</td>
</tr>
<tr>
<td>DECIMAL(p,s)</td>
<td>DECIMAL(p,s)</td>
<td></td>
</tr>
<tr>
<td>IBM DB2 LUW data type</td>
<td>SQL Anywhere or UltraLite data type</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------------------</td>
<td>----------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>GRAPHIC((n))</td>
<td>VARCHAR((2n))</td>
<td>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.</td>
</tr>
<tr>
<td>INT</td>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td>VARCHAR((32700))</td>
<td></td>
</tr>
<tr>
<td>LONG VARCHAR FOR BIT DATA</td>
<td>VARBINARY((32700))</td>
<td></td>
</tr>
<tr>
<td>LONG VARGRAPHIC((n))</td>
<td>VARCHAR((32700))</td>
<td>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.</td>
</tr>
<tr>
<td>NUMERIC((p,s))</td>
<td>NUMERIC((p,s))</td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>REAL can cause problems if the consolidated and remote databases don't allow the exact same (imprecise) values. Not all possible values are tested, so care must be taken. To avoid problems, do not use these types as part of a primary key.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SMALLINT</td>
<td></td>
</tr>
</tbody>
</table>
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. 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.

| IBM DB2 LUW data type | SQL Anywhere or UltraLite data type | Notes |
|---|---|---
| TIMESTAMP | TIMESTAMP | |
| VARCHAR(n) | VARCHAR(n) | |
| VARCHAR(n) FOR BIT DATA | VARBINARY(n) | |
| VARGRAPHIC(n) | VARCHAR(2n) | 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 DATETIME on the remote database should be type DATETIME2 on the consolidated database.

<table>
<thead>
<tr>
<th>SQL Anywhere or UltraLite data type</th>
<th>Microsoft SQL Server data type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>BIGINT</td>
<td></td>
</tr>
<tr>
<td>BINARY(n&lt;=8000)</td>
<td>VARBINARY(n)</td>
<td></td>
</tr>
<tr>
<td>BINARY(n&gt;8000)</td>
<td>VARBINARY(MAX)</td>
<td></td>
</tr>
<tr>
<td>SQL Anywhere or UltraLite data type</td>
<td>Microsoft SQL Server data type</td>
<td>Notes</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>---------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>BIT</td>
<td>BIT</td>
<td></td>
</tr>
<tr>
<td>CHAR(n&lt;=8000)</td>
<td>VARCHAR(n)</td>
<td></td>
</tr>
<tr>
<td>CHAR(n&gt;8000)</td>
<td>VARCHAR(MAX)</td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td>DATETIME</td>
<td>DATETIME2</td>
<td>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.</td>
</tr>
<tr>
<td>DECIMAL(p&lt;=38,s)</td>
<td>DECIMAL(p,s)</td>
<td>Microsoft SQL Server DECIMAL/NUMERIC precision ranges from 1 to 38, so p must be less than 39.</td>
</tr>
<tr>
<td>DECIMAL(p&gt;38,s)</td>
<td></td>
<td>There is no corresponding data type in Microsoft SQL Server.</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>FLOAT(53)</td>
<td></td>
</tr>
<tr>
<td>FLOAT(p)</td>
<td>FLOAT(p)</td>
<td></td>
</tr>
<tr>
<td>IMAGE</td>
<td>VARBINARY(MAX)</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>LONG BINARY</td>
<td>VARBINARY(MAX)</td>
<td></td>
</tr>
<tr>
<td>LONG NVARCHAR</td>
<td>NVARCHAR(MAX)</td>
<td></td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td>VARCHAR(MAX)</td>
<td></td>
</tr>
<tr>
<td>LONG VARBIT</td>
<td>VARCHAR(MAX)</td>
<td></td>
</tr>
<tr>
<td>MONEY</td>
<td>MONEY</td>
<td></td>
</tr>
<tr>
<td>NCHAR(n&lt;=4000)</td>
<td>NVARCHAR(c)</td>
<td></td>
</tr>
<tr>
<td>NCHAR(n&gt;4000)</td>
<td>NVARCHAR(MAX)</td>
<td></td>
</tr>
<tr>
<td>SQL Anywhere or UltraLite data type</td>
<td>Microsoft SQL Server data type</td>
<td>Notes</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>---------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>NTEXT</td>
<td>NVARCHAR(MAX)</td>
<td></td>
</tr>
<tr>
<td>NUMERIC($p$&lt;38,$s$)</td>
<td>NUMERIC($p$,$s$)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NUMERIC($p$&gt;38,$s$)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NVARCHAR($n$$\leq$$4000$)</td>
<td>NVARCHAR($c$)</td>
<td></td>
</tr>
<tr>
<td>NVARCHAR($n$$&gt;$$4000$)</td>
<td>NVARCHAR(MAX)</td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>SMALLDATETIME</td>
<td>SMALLDATETIME</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SMALLMONEY</td>
<td>SMALLMONEY</td>
<td></td>
</tr>
<tr>
<td>ST_GEOMETRY</td>
<td>GEOMETRY</td>
<td></td>
</tr>
<tr>
<td>TEXT</td>
<td>VARCHAR(MAX)</td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>TIME</td>
<td></td>
</tr>
</tbody>
</table>

Microsoft SQL Server DECIMAL/NUMERIC precision ranges from 1 to 38, so $p$ must be less than 39.

There is no corresponding data type in Microsoft SQL Server.

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.
<table>
<thead>
<tr>
<th>SQL Anywhere or UltraLite data type</th>
<th>Microsoft SQL Server data type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMESTAMP</td>
<td>DATETIME2</td>
<td>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.</td>
</tr>
<tr>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>DATETIMEOFFSET</td>
<td></td>
</tr>
<tr>
<td>TINYINT</td>
<td>TINYINT</td>
<td>For download, values must be non-negative.</td>
</tr>
<tr>
<td>UNIQUEIDENTIFIER</td>
<td>UNIQUEIDENTIFIER</td>
<td></td>
</tr>
<tr>
<td>UNIQUEIDENTIFIERSTR</td>
<td>UNIQUEIDENTIFIER</td>
<td></td>
</tr>
<tr>
<td>UNSIGNED BIGINT</td>
<td>NUMERIC(20)</td>
<td>For download, values must be non-negative.</td>
</tr>
<tr>
<td>UNSIGNED INTEGER</td>
<td>NUMERIC(11)</td>
<td>For download, values must be non-negative.</td>
</tr>
<tr>
<td>UNSIGNED TINYINT</td>
<td>TINYINT</td>
<td>For download, values must be non-negative.</td>
</tr>
<tr>
<td>UNSIGNED SMALLINT</td>
<td>INT</td>
<td>For download, values must be non-negative.</td>
</tr>
<tr>
<td>VARBINARY(n&lt;=8000)</td>
<td>VARBINARY(n)</td>
<td></td>
</tr>
<tr>
<td>VARBINARY(n&gt;8000)</td>
<td>VARBINARY(MAX)</td>
<td></td>
</tr>
<tr>
<td>VARBIT(n&lt;=8000)</td>
<td>VARCHAR(n)</td>
<td></td>
</tr>
<tr>
<td>VARBIT(n&gt;8000)</td>
<td>VARCHAR(MAX)</td>
<td></td>
</tr>
<tr>
<td>VARCHAR(n&lt;=8000)</td>
<td>VARCHAR(c)</td>
<td></td>
</tr>
<tr>
<td>VARCHAR(n&gt;8000)</td>
<td>VARCHAR(MAX)</td>
<td></td>
</tr>
<tr>
<td>XML</td>
<td>XML or VARCHAR(MAX)</td>
<td>For Microsoft SQL Server 2005, use XML. For other versions, use VARCHAR(MAX).</td>
</tr>
</tbody>
</table>
### 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.

<table>
<thead>
<tr>
<th>Microsoft SQL Server data type</th>
<th>SQL Anywhere or UltraLite data type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>BIGINT</td>
<td></td>
</tr>
<tr>
<td>BINARY(n)</td>
<td>BINARY(n)</td>
<td></td>
</tr>
<tr>
<td>BIT</td>
<td>BIT</td>
<td></td>
</tr>
<tr>
<td>CHAR(n)</td>
<td>VARCHAR(n)</td>
<td>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.</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td>DATETIME</td>
<td>TIMESTAMP or DATETIME</td>
<td>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.</td>
</tr>
</tbody>
</table>

---

---

---
<table>
<thead>
<tr>
<th>Microsoft SQL Server data type</th>
<th>SQL Anywhere or UltraLite data type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATETIME2</td>
<td>TIMESTAMP</td>
<td>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.</td>
</tr>
<tr>
<td>DECIMAL(p,s)</td>
<td>DECIMAL(p,s)</td>
<td></td>
</tr>
<tr>
<td>FLOAT(p)</td>
<td>FLOAT(p)</td>
<td></td>
</tr>
<tr>
<td>GEOMETRY</td>
<td>ST_GEOMETRY</td>
<td></td>
</tr>
<tr>
<td>IMAGE</td>
<td>LONG BINARY</td>
<td></td>
</tr>
<tr>
<td>INT</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>MONEY</td>
<td>MONEY</td>
<td></td>
</tr>
<tr>
<td>NCHAR(n)</td>
<td>NVARCHAR(c)</td>
<td>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.</td>
</tr>
<tr>
<td>NTEXT</td>
<td>LONG NVARCHAR</td>
<td>Not available in UltraLite.</td>
</tr>
<tr>
<td>NVARCHAR(c)</td>
<td>NVARCHAR(c)</td>
<td>Not available in UltraLite.</td>
</tr>
<tr>
<td>NVARCHAR(MAX)</td>
<td>LONG NVARCHAR</td>
<td>Not available in UltraLite.</td>
</tr>
<tr>
<td>NUMERIC(p,s)</td>
<td>NUMERIC(p,s)</td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>REAL can cause problems if the consolidated and remote databases don't allow the exact same (imprecise) values. Not all possible values are tested, so care must be taken. To avoid problems, do not use these types as part of a primary key.</td>
</tr>
<tr>
<td>Microsoft SQL Server data type</td>
<td>SQL Anywhere or UltraLite data type</td>
<td>Notes</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td>SMALLDATETIME</td>
<td>SMALLDATETIME</td>
<td>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.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SMALLINT</td>
<td></td>
</tr>
<tr>
<td>SMALLMONEY</td>
<td>SMALLMONEY</td>
<td></td>
</tr>
<tr>
<td>TEXT</td>
<td>LONG VARCHAR</td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>TIME</td>
<td>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.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>VARBINARY(8)</td>
<td>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.</td>
</tr>
<tr>
<td>DATETIMEOFFSET</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td></td>
</tr>
</tbody>
</table>
### 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.

<table>
<thead>
<tr>
<th>SQL Anywhere or UltraLite data type</th>
<th>MySQL data type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>BIGINT</td>
<td></td>
</tr>
<tr>
<td>BINARY((n &lt;= 255))</td>
<td>BINARY(n)</td>
<td></td>
</tr>
<tr>
<td>BINARY((n &gt; 255))</td>
<td>BLOB</td>
<td></td>
</tr>
<tr>
<td>BIT</td>
<td>BIT</td>
<td></td>
</tr>
<tr>
<td>CHAR((n &lt;= 255))</td>
<td>CHAR(n)</td>
<td></td>
</tr>
<tr>
<td>CHAR((n &gt; 255))</td>
<td>TEXT(n)</td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
<td>The year must range from 1000 to 9999.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>DATETIME</td>
<td>The MySQL DATETIME data type does not support fractional seconds. The year must range from 1000 to 9999.</td>
</tr>
<tr>
<td>DECIMAL((p &lt;= 65, s &lt;= 30))</td>
<td>DECIMAL(p, s)</td>
<td></td>
</tr>
<tr>
<td>SQL Anywhere or UltraLite data type</td>
<td>MySQL data type</td>
<td>Notes</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-----------------</td>
<td>-------</td>
</tr>
<tr>
<td>DECIMAL($p&gt;65, s&gt;30$)</td>
<td></td>
<td>There is no corresponding data type in MySQL if the precision is greater than 65 or if the scale is greater than 30.</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td></td>
</tr>
<tr>
<td>FLOAT</td>
<td>FLOAT</td>
<td></td>
</tr>
<tr>
<td>IMAGE</td>
<td>LONGBLOB</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>LONG BINARY</td>
<td>LONGBLOB</td>
<td></td>
</tr>
<tr>
<td>LONG NVARCHAR</td>
<td>LONGTEXT CHARACTER SET UTF8</td>
<td></td>
</tr>
<tr>
<td>LONG VARBIT</td>
<td>LONGTEXT</td>
<td></td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td>LONGTEXT</td>
<td></td>
</tr>
<tr>
<td>MONEY</td>
<td>NUMERIC(19,4)</td>
<td></td>
</tr>
<tr>
<td>NCHAR($n\leq255$)</td>
<td>CHAR($n$) CHARACTER SET UTF8</td>
<td></td>
</tr>
<tr>
<td>NCHAR($n&gt;255$)</td>
<td>TEXT CHARACTER SET UTF8</td>
<td></td>
</tr>
<tr>
<td>NTEXT</td>
<td>LONGTEXT CHARACTER SET UTF8</td>
<td></td>
</tr>
<tr>
<td>NUMERIC($p\leq65, s\leq30$)</td>
<td>DECIMAL($p, s$)</td>
<td></td>
</tr>
<tr>
<td>NUMERIC($p&gt;65, s&gt;30$)</td>
<td></td>
<td>There is no corresponding data type in MySQL.</td>
</tr>
<tr>
<td>NVARCHAR($n$)</td>
<td>VARCHAR($n$) CHARACTER SET UTF8</td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>SMALLDATETIME</td>
<td>DATETIME</td>
<td>The MySQL DATETIME data type does not support fractional seconds. The year must range from 1000 to 9999.</td>
</tr>
<tr>
<td>SQL Anywhere or UltraLite data type</td>
<td>MySQL data type</td>
<td>Notes</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>----------------</td>
<td>-------</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SMALLINT</td>
<td></td>
</tr>
<tr>
<td>SMALLMONEY</td>
<td>NUMERIC(10,4)</td>
<td></td>
</tr>
<tr>
<td>ST_GEOMETRY</td>
<td>GEOMETRY</td>
<td></td>
</tr>
<tr>
<td>TEXT</td>
<td>LONGTEXT</td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>TIME</td>
<td>The MySQL TIME data type does not support fractional seconds.</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>DATETIME</td>
<td>The MySQL DATETIME data type does not support fractional seconds. The year must range from 1000 to 9999.</td>
</tr>
<tr>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>VARCHAR(34)</td>
<td>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.</td>
</tr>
<tr>
<td>TINYINT</td>
<td>TINYINT UNSIGNED</td>
<td>TINYINT is always unsigned in SQL Anywhere and UltraLite.</td>
</tr>
<tr>
<td>UNIQUEIDENTIFIER</td>
<td>CHAR(36)</td>
<td></td>
</tr>
<tr>
<td>UNIQUEIDENTIFIERSTR</td>
<td>CHAR(36)</td>
<td></td>
</tr>
<tr>
<td>VARBINARY(n)</td>
<td>VARCHAR(n)</td>
<td></td>
</tr>
<tr>
<td>VARBIT(n&lt;=8000)</td>
<td>VARCHAR(n)</td>
<td></td>
</tr>
<tr>
<td>VARBIT(n&gt;8000)</td>
<td>TEXT</td>
<td></td>
</tr>
<tr>
<td>VARCHAR(n)</td>
<td>VARCHAR(n)</td>
<td></td>
</tr>
<tr>
<td>XML</td>
<td>LONGTEXT</td>
<td></td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>MySQL data type</th>
<th>SQL Anywhere or UltraLite data type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>BIGINT</td>
<td></td>
</tr>
<tr>
<td>BINARY(n)</td>
<td>BINARY(n)</td>
<td></td>
</tr>
<tr>
<td>BIT(1)</td>
<td>BIT</td>
<td></td>
</tr>
<tr>
<td>BIT(n&gt;1)</td>
<td>UNSIGNED BIGINT</td>
<td></td>
</tr>
<tr>
<td>BLOB(n&lt;=32767)</td>
<td>VARBINARY(n)</td>
<td></td>
</tr>
<tr>
<td>BLOB(n&gt;32767)</td>
<td>IMAGE</td>
<td></td>
</tr>
<tr>
<td>BOOL</td>
<td>BIT</td>
<td></td>
</tr>
<tr>
<td>CHAR(n)</td>
<td>CHAR(n)</td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
<td>The year must range from 1000 to 9999.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>DATETIME</td>
<td>The MySQL DATETIME data type does not support fractional seconds. The year must range from 1000 to 9999.</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td></td>
</tr>
<tr>
<td>DECIMAL</td>
<td>DECIMAL</td>
<td></td>
</tr>
<tr>
<td>ENUM</td>
<td></td>
<td>There is no corresponding data type in SQL Anywhere or UltraLite.</td>
</tr>
<tr>
<td>GEOMETRY</td>
<td>ST_GEOMETRY</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>LINestring</td>
<td></td>
<td>There is no corresponding data type in SQL Anywhere or UltraLite.</td>
</tr>
<tr>
<td>LONGBLOB</td>
<td>IMAGE</td>
<td></td>
</tr>
<tr>
<td>LONGTEXT</td>
<td>TEXT</td>
<td></td>
</tr>
<tr>
<td>MEDIUMBLOB</td>
<td>IMAGE</td>
<td></td>
</tr>
<tr>
<td>MySQL data type</td>
<td>SQL Anywhere or UltraLite data type</td>
<td>Notes</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------</td>
<td>-----------------------------------------------------------------------</td>
</tr>
<tr>
<td>MEDIUMINT</td>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>MEDIUMTEXT</td>
<td>TEXT</td>
<td></td>
</tr>
<tr>
<td>MULTILINestring</td>
<td></td>
<td>There is no corresponding data type in SQL Anywhere or UltraLite.</td>
</tr>
<tr>
<td>MULTIPoiNT</td>
<td></td>
<td>There is no corresponding data type in SQL Anywhere or UltraLite.</td>
</tr>
<tr>
<td>MULTIPoLYGON</td>
<td></td>
<td>There is no corresponding data type in SQL Anywhere or UltraLite.</td>
</tr>
<tr>
<td>NCHAR</td>
<td>NCHAR</td>
<td>Not available in UltraLite.</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>NUMERIC</td>
<td></td>
</tr>
<tr>
<td>NVARCHAR</td>
<td>NVARCHAR</td>
<td>Not available in UltraLite.</td>
</tr>
<tr>
<td>POINT</td>
<td></td>
<td>There is no corresponding data type in SQL Anywhere or UltraLite.</td>
</tr>
<tr>
<td>POLYGON</td>
<td></td>
<td>There is no corresponding data type in SQL Anywhere or UltraLite.</td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>SET</td>
<td></td>
<td>There is no corresponding data type in SQL Anywhere or UltraLite.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SMALLINT</td>
<td></td>
</tr>
<tr>
<td>TEXT(n&lt;=32767)</td>
<td>VARCHAR(n)</td>
<td></td>
</tr>
<tr>
<td>TEXT(n&gt;32767)</td>
<td>TEXT</td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>TIME</td>
<td>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'.</td>
</tr>
</tbody>
</table>
### MySQL data types

<table>
<thead>
<tr>
<th>MySQL data type</th>
<th>SQL Anywhere or UltraLite data type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMESTAMP</td>
<td>TIMESTAMP</td>
<td>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.</td>
</tr>
<tr>
<td>TINYBLOB</td>
<td>VARBINARY</td>
<td></td>
</tr>
<tr>
<td>TINYINT</td>
<td>SMALLINT</td>
<td>TINYINT is always unsigned in SQL Anywhere and UltraLite. Must be a positive value.</td>
</tr>
<tr>
<td>TINYINT UNSIGNED</td>
<td>TINYINT</td>
<td>TINYINT is always unsigned in SQL Anywhere and UltraLite.</td>
</tr>
<tr>
<td>TINYTEXT</td>
<td>VARCHAR</td>
<td></td>
</tr>
<tr>
<td>VARBINARY(n&lt;=32767)</td>
<td>VARBINARY(n)</td>
<td></td>
</tr>
<tr>
<td>VARBINARY(n&gt;32767)</td>
<td>IMAGE</td>
<td></td>
</tr>
<tr>
<td>VARCHAR(n&lt;=32767)</td>
<td>VARCHAR(n)</td>
<td></td>
</tr>
<tr>
<td>VARCHAR(n&gt;32767)</td>
<td>TEXT</td>
<td></td>
</tr>
<tr>
<td>YEAR[(2</td>
<td>4)]</td>
<td>INTEGER</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>SQL Anywhere or UltraLite data type</th>
<th>Oracle data type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>NUMBER(20)</td>
<td></td>
</tr>
<tr>
<td>SQL Anywhere or UltraLite data type</td>
<td>Oracle data type</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------</td>
<td>-------</td>
</tr>
<tr>
<td>BINARY($n \leq 2000$)</td>
<td>RAW($n$)</td>
<td></td>
</tr>
<tr>
<td>BINARY($n &gt; 2000$)</td>
<td>BLOB</td>
<td>Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.</td>
</tr>
<tr>
<td>BIT</td>
<td>NUMBER(1)</td>
<td></td>
</tr>
<tr>
<td>CHAR($n \leq 4000$)</td>
<td>VARCHAR2($n$ byte)</td>
<td>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.</td>
</tr>
<tr>
<td>CHAR($n &gt; 4000$)</td>
<td>CLOB</td>
<td>Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE or TIME-STAMP</td>
<td>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.</td>
</tr>
<tr>
<td>DATETIME</td>
<td>DATE or TIME-STAMP</td>
<td>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.</td>
</tr>
<tr>
<td>SQL Anywhere or UltraLite data type</td>
<td>Oracle data type</td>
<td>Notes</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>-----------------</td>
<td>-------</td>
</tr>
<tr>
<td>DECIMAL(p\leq38,s)</td>
<td>NUMBER(p, 0\leq s\leq38)</td>
<td>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.</td>
</tr>
<tr>
<td>DECIMAL(p&gt;38,s)</td>
<td></td>
<td>There is no corresponding data type in Oracle.</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE PRECISION or BINARY_DOUBLE(^1)</td>
<td>The special values INF, -INF and NAN of Oracle Database 10g BINARY_FLOAT and BINARY_DOUBLE cannot be synchronized with SQL Anywhere or UltraLite.</td>
</tr>
<tr>
<td>FLOAT(p)</td>
<td>FLOAT(p)</td>
<td></td>
</tr>
<tr>
<td>IMAGE</td>
<td>BLOB</td>
<td>Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.</td>
</tr>
<tr>
<td>INTEGER</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>LONG BINARY</td>
<td>BLOB</td>
<td>Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.</td>
</tr>
<tr>
<td>LONG NVARCHAR</td>
<td>NCLOB</td>
<td>Oracle CLOB and NCLOB can hold up to 4G of data. SQL Anywhere LONG VARCHAR and LONG NVARCHAR can only hold up to 2G.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.</td>
</tr>
<tr>
<td>LONG VARBIT</td>
<td>CLOB</td>
<td>Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.</td>
</tr>
<tr>
<td>SQL Anywhere or UltraLite data type</td>
<td>Oracle data type</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------</td>
<td>-------</td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td>CLOB</td>
<td>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.</td>
</tr>
<tr>
<td>MONEY</td>
<td>NUMBER(19,4)</td>
<td></td>
</tr>
<tr>
<td>NCHAR(c)</td>
<td>NVARCHAR2(c char) or NCLOB</td>
<td>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.</td>
</tr>
<tr>
<td>NTEXT</td>
<td>NCLOB</td>
<td>Oracle NCLOB can hold up to 4G of data. SQL Anywhere NTEXT (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.</td>
</tr>
<tr>
<td>NUMERIC(p&lt;=38,s)</td>
<td>NUMBER(p, 0&lt;=s&lt;=38)</td>
<td>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.</td>
</tr>
<tr>
<td>NUMERIC(p&gt;38,s)</td>
<td></td>
<td>There is no corresponding data type in Oracle.</td>
</tr>
<tr>
<td>SQL Anywhere or UltraLite data type</td>
<td>Oracle data type</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>------------------</td>
<td>-------</td>
</tr>
<tr>
<td>NVARCHAR</td>
<td>NVARCHAR2(c char) or NCLOB</td>
<td>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.</td>
</tr>
<tr>
<td>REAL</td>
<td>REAL or BINARY_FLOAT¹</td>
<td>The special values INF, -INF and NAN of Oracle Database 10g BINARY_FLOAT and BINARY_DOUBLE cannot be synchronized with SQL Anywhere or UltraLite.</td>
</tr>
<tr>
<td>SMALLDATETIME</td>
<td>DATE² or TIME-STAMP</td>
<td>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.</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>NUMBER(5)</td>
<td></td>
</tr>
<tr>
<td>SMALLMONEY</td>
<td>NUMBER(10,4)</td>
<td></td>
</tr>
<tr>
<td>ST_GEOMETRY</td>
<td>SDO_GEOMETRY</td>
<td></td>
</tr>
<tr>
<td>TEXT</td>
<td>CLOB</td>
<td>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.</td>
</tr>
<tr>
<td>TIME</td>
<td>DATE² or TIME-STAMP</td>
<td>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.</td>
</tr>
<tr>
<td>SQL Anywhere or UltraLite data type</td>
<td>Oracle data type</td>
<td>Notes</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>------------------</td>
<td>-------</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>DATE or TIMESTAMP</td>
<td>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.</td>
</tr>
<tr>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>TIMESTAMP WITH TIME ZONE</td>
<td></td>
</tr>
<tr>
<td>TINYINT</td>
<td>NUMBER(3)</td>
<td>For download, Oracle values must be non-negative.</td>
</tr>
<tr>
<td>UNSIGNED BIGINT</td>
<td>NUMBER(20)</td>
<td>For download, Oracle values must be non-negative.</td>
</tr>
<tr>
<td>UNSIGNED INTEGER</td>
<td>NUMBER(11)</td>
<td>For download, Oracle values must be non-negative.</td>
</tr>
<tr>
<td>UNSIGNED SMALLINT</td>
<td>NUMBER(5)</td>
<td>For download, Oracle values must be non-negative.</td>
</tr>
<tr>
<td>UNSIGNED TINYINT</td>
<td>NUMBER(3)</td>
<td>For download, Oracle values must be non-negative.</td>
</tr>
<tr>
<td>UNIQUEIDENTIFIER</td>
<td>CHAR(36)</td>
<td></td>
</tr>
<tr>
<td>UNIQUEIDENTIFIERSTR</td>
<td>CHAR(36)</td>
<td>UNIQUEIDENTIFIERSTR is not recommended to use for Oracle. Use UNIQUEIDENTIFIER instead.</td>
</tr>
<tr>
<td>VARBINARY(n \leq 2000)</td>
<td>RAW(n)</td>
<td></td>
</tr>
<tr>
<td>VARBINARY(n &gt; 2000)</td>
<td>BLOB</td>
<td>Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.</td>
</tr>
<tr>
<td>VARBIT(n \leq 4000)</td>
<td>VARCHAR2(n)</td>
<td></td>
</tr>
<tr>
<td>VARBIT(n &gt; 4000)</td>
<td>CLOB</td>
<td>Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.</td>
</tr>
</tbody>
</table>
### 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.

<table>
<thead>
<tr>
<th>Oracle data type</th>
<th>SQL Anywhere or UltraLite data type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFILE</td>
<td>LONG BINARY</td>
<td>Download only. Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.</td>
</tr>
<tr>
<td>Oracle data type</td>
<td>SQL Anywhere or UltraLite data type</td>
<td>Notes</td>
</tr>
<tr>
<td>------------------------</td>
<td>-------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>BINARY_DOUBLE</td>
<td>DOUBLE</td>
<td>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.</td>
</tr>
<tr>
<td>BINARY_FLOAT</td>
<td>REAL</td>
<td>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.</td>
</tr>
<tr>
<td>BLOB</td>
<td>LONG BINARY</td>
<td>Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.</td>
</tr>
<tr>
<td>CHAR(n byte)</td>
<td>VARCHAR(n)</td>
<td>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.</td>
</tr>
<tr>
<td>CLOB</td>
<td>LONG VARCHAR</td>
<td>Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.</td>
</tr>
<tr>
<td>DATE</td>
<td>TIMESTAMP</td>
<td>The year must be in the range 1-9999.</td>
</tr>
<tr>
<td>INTERVAL YEAR(year_precision) TO MONTH</td>
<td></td>
<td>There is no corresponding data type in SQL Anywhere or UltraLite.</td>
</tr>
<tr>
<td>INTERVAL DAY(day_precision) TO SECOND(p)</td>
<td></td>
<td>There is no corresponding data type in SQL Anywhere or UltraLite.</td>
</tr>
<tr>
<td>Oracle data type</td>
<td>SQL Anywhere or UltraLite data type</td>
<td>Notes</td>
</tr>
<tr>
<td>--------------------</td>
<td>------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LONG</td>
<td>LONG VARCHAR</td>
<td></td>
</tr>
<tr>
<td>LONG RAW</td>
<td>LONG BINARY</td>
<td></td>
</tr>
<tr>
<td>NCHAR(c char)</td>
<td>NVARCHAR(c)</td>
<td>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.</td>
</tr>
<tr>
<td>NCLOB</td>
<td>LONG NVARCHAR</td>
<td>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.</td>
</tr>
<tr>
<td>NUMBER(p,s)</td>
<td>NUMBER(p,s)</td>
<td>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.</td>
</tr>
<tr>
<td>VARCHAR2(c char)</td>
<td>NVARCHAR(c)</td>
<td>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.</td>
</tr>
<tr>
<td>RAW</td>
<td>BINARY</td>
<td>SQL Anywhere or UltraLite values can be longer than Oracle values, so make sure values are not too big when uploading.</td>
</tr>
<tr>
<td>ROWID</td>
<td>VARCHAR(64)</td>
<td>UROWID and ROWID are read-only and so are unlikely to be synchronized.</td>
</tr>
<tr>
<td>SDO_GEOMETRY</td>
<td>ST_GEOMETRY</td>
<td></td>
</tr>
</tbody>
</table>
### SAP HANA database server data type mapping

**Mapping to SAP HANA consolidated data types**

The following table identifies how SQL Anywhere and UltraLite remote data types are mapped to SAP HANA consolidated data types. For example, a column of type LONG VARBIT on the remote database should be type CLOB on the consolidated database.

<table>
<thead>
<tr>
<th>SQL Anywhere or UltraLite data type</th>
<th>SAP HANA data type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT</td>
<td>TINYINT</td>
<td></td>
</tr>
<tr>
<td>TINYINT</td>
<td>TINYINT</td>
<td></td>
</tr>
<tr>
<td>SQL Anywhere or UltraLite data type</td>
<td>SAP HANA data type</td>
<td>Notes</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>--------------------</td>
<td>-------</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SMALLINT</td>
<td></td>
</tr>
<tr>
<td>UNSIGNED SMALLINT</td>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>UNSIGNED INTEGER</td>
<td>BIGINT</td>
<td></td>
</tr>
<tr>
<td>BIGINT</td>
<td>BIGINT</td>
<td></td>
</tr>
<tr>
<td>UNSIGNED BIGINT</td>
<td>DECIMAL(20,0)</td>
<td></td>
</tr>
<tr>
<td>DECIMAL((p,s))</td>
<td>DECIMAL((p,s))</td>
<td></td>
</tr>
<tr>
<td>NUMERIC((p,s))</td>
<td>DECIMAL((p,s))</td>
<td></td>
</tr>
<tr>
<td>FLOAT</td>
<td>FLOAT</td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td></td>
</tr>
<tr>
<td>SMALLMONEY</td>
<td>DECIMAL((10,4))</td>
<td></td>
</tr>
<tr>
<td>MONEY</td>
<td>DECIMAL((19,4))</td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>TIME</td>
<td>SQL Anywhere and UltraLite fractional seconds cannot be preserved when using an SAP HANA TIME data type, which has no fractional seconds. To avoid problems, do not use fractional seconds.</td>
</tr>
<tr>
<td>SMALLDATETIME</td>
<td>TIMESTAMP</td>
<td></td>
</tr>
<tr>
<td>DATETIME</td>
<td>TIMESTAMP</td>
<td></td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>TIMESTAMP</td>
<td></td>
</tr>
<tr>
<td>SQL Anywhere or UltraLite data type</td>
<td>SAP HANA data type</td>
<td>Notes</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>------------------</td>
<td>-------</td>
</tr>
<tr>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>VARCHAR(34)</td>
<td>There is no equivalent data type in SAP HANA so a TIMESTAMP WITH TIME ZONE column should be mapped to a VARCHAR(34) column. During upload, the MobiLink server converts the data to a string using the format 'yyyy-mm-dd hh:mm:ss.ssssss [+</td>
</tr>
<tr>
<td>CHAR(n&lt;=5000)</td>
<td>VARCHAR(n)</td>
<td></td>
</tr>
<tr>
<td>CHAR(n&gt;5000)</td>
<td>CLOB</td>
<td></td>
</tr>
<tr>
<td>VARCHAR(n&lt;=5000)</td>
<td>VARCHAR(n)</td>
<td></td>
</tr>
<tr>
<td>VARCHAR(n&gt;5000)</td>
<td>CLOB</td>
<td></td>
</tr>
<tr>
<td>LONG VARCHAR</td>
<td>CLOB</td>
<td></td>
</tr>
<tr>
<td>NCHAR(n&lt;=5000)</td>
<td>NVARCHAR</td>
<td></td>
</tr>
<tr>
<td>NCHAR(n&gt;5000)</td>
<td>NCLOB</td>
<td></td>
</tr>
<tr>
<td>NVARCHAR(n&lt;=5000)</td>
<td>NVARCHAR(n)</td>
<td></td>
</tr>
<tr>
<td>NVARCHAR(n&gt;5000)</td>
<td>NCLOB</td>
<td></td>
</tr>
<tr>
<td>LONG NVARCHAR</td>
<td>NCLOB</td>
<td></td>
</tr>
<tr>
<td>BINARY(n&lt;=5000)</td>
<td>VARBINARY(n)</td>
<td></td>
</tr>
<tr>
<td>BINARY(n&gt;5000)</td>
<td>BLOB</td>
<td></td>
</tr>
<tr>
<td>VARBINARY(n&lt;=5000)</td>
<td>VARBINARY(n)</td>
<td></td>
</tr>
<tr>
<td>VARBINARY(n&gt;5000)</td>
<td>BLOB</td>
<td></td>
</tr>
<tr>
<td>LONG BINARY</td>
<td>BLOB</td>
<td></td>
</tr>
<tr>
<td>VARBIT(n&lt;=5000)</td>
<td>VARCHAR(n)</td>
<td></td>
</tr>
</tbody>
</table>
### Mapping to SQL Anywhere or UltraLite remote data types

The following table identifies how SAP HANA consolidated data types are mapped to SQL Anywhere and UltraLite remote data types. For example, a column of type ALPHANUM($n$) on the consolidated database should be type VARCHAR($n$) on the remote database.

<table>
<thead>
<tr>
<th>SAP HANA data type</th>
<th>SQL Anywhere or UltraLite data type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>TINYINT</td>
<td>TINYINT</td>
<td></td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SMALLINT</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>INTEGER</td>
<td></td>
</tr>
<tr>
<td>BIGINT</td>
<td>BIGINT</td>
<td></td>
</tr>
<tr>
<td>SMALLDECIMAL</td>
<td>DECIMAL($p,s$)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>In SQL Anywhere, DECIMAL $p$ is between 1 and 127 and $s$ is always less than or equal to $p$. In SAP HANA, $p$ ranges from 1 to 16 and $s$ ranges from -369 to 368.</td>
</tr>
<tr>
<td>DECIMAL($p,s$)</td>
<td>DECIMAL($p,s$)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>In SQL Anywhere, DECIMAL $p$ is between 1 and 127 and $s$ is always less than or equal to $p$. In SAP HANA, $p$ ranges from 1 to 34 and $s$ ranges from -6111 to 6176.</td>
</tr>
<tr>
<td>FLOAT</td>
<td>DOUBLE</td>
<td>SAP HANA promotes FLOAT and REAL to DOUBLE.</td>
</tr>
<tr>
<td>REAL</td>
<td>DOUBLE</td>
<td>SAP HANA promotes FLOAT and REAL to DOUBLE.</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
<td></td>
</tr>
</tbody>
</table>
### SAP Sybase IQ Enterprise data mapping

**Mapping to SAP Sybase IQ consolidated data types**

The following table identifies how SQL Anywhere and UltraLite remote data types are mapped to SAP Sybase IQ consolidated data types. For example, a column of type LONG VARBIT on the remote database should be type LONG VARCHAR on the consolidated database.

<table>
<thead>
<tr>
<th>SQL Anywhere or UltraLite data type</th>
<th>SAP Sybase IQ</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>BIGINT</td>
<td></td>
</tr>
<tr>
<td>BIT</td>
<td>BIT</td>
<td></td>
</tr>
<tr>
<td>BINARY(n)</td>
<td>BINARY(n)</td>
<td></td>
</tr>
<tr>
<td>SQL Anywhere or UltraLite data type</td>
<td>SAP Sybase IQ</td>
<td>Notes</td>
</tr>
<tr>
<td>-------------------------------------</td>
<td>--------------</td>
<td>-------</td>
</tr>
<tr>
<td>CHAR(n)</td>
<td>CHAR(n)</td>
<td>There are some restrictions on CHAR and VARCHAR columns over 255 bytes. For more information, see the SAP Sybase IQ documentation.</td>
</tr>
<tr>
<td>DATE</td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td>DATETIME</td>
<td>DATETIME</td>
<td></td>
</tr>
<tr>
<td>DECIMAL(p,s)</td>
<td>DECIMAL(p,s)</td>
<td></td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td></td>
</tr>
<tr>
<td>FLOAT(p)</td>
<td>FLOAT(p)</td>
<td></td>
</tr>
<tr>
<td>INT</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>LONG BINARY / IMAGE</td>
<td>LONG BINARY / IMAGE</td>
<td></td>
</tr>
<tr>
<td>LONG NVARCHAR / NTEXT</td>
<td></td>
<td>This data type is not available in SAP Sybase IQ.</td>
</tr>
<tr>
<td>LONG VARBIT</td>
<td>LONG VARCHAR</td>
<td></td>
</tr>
<tr>
<td>LONG VARCHAR / TEXT</td>
<td>TEXT</td>
<td></td>
</tr>
<tr>
<td>MONEY</td>
<td>MONEY</td>
<td></td>
</tr>
<tr>
<td>NCHAR(n)</td>
<td></td>
<td>This data type is not available in SAP Sybase IQ.</td>
</tr>
<tr>
<td>NVARCHAR(n)</td>
<td></td>
<td>This data type is not available in SAP Sybase IQ.</td>
</tr>
<tr>
<td>NUMERIC(p,s)</td>
<td>NUMERIC(p,s)</td>
<td></td>
</tr>
<tr>
<td>SMALLDATETIME</td>
<td>SMALLDATETIME</td>
<td></td>
</tr>
<tr>
<td>SMALLMONEY</td>
<td>SMALLMONEY</td>
<td></td>
</tr>
<tr>
<td>ST_GEOMETRY</td>
<td></td>
<td>This data type is not available in SAP Sybase IQ.</td>
</tr>
<tr>
<td>TIME</td>
<td>TIME</td>
<td></td>
</tr>
<tr>
<td>SQL Anywhere or UltraLite data type</td>
<td>SAP Sybase IQ</td>
<td>Notes</td>
</tr>
<tr>
<td>------------------------------------</td>
<td>--------------</td>
<td>-------</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>TIMESTAMP</td>
<td></td>
</tr>
<tr>
<td>TIMESTAMP WITH TIME ZONE</td>
<td>VARCHAR(34)</td>
<td></td>
</tr>
<tr>
<td>TINYINT</td>
<td>TINYINT</td>
<td></td>
</tr>
<tr>
<td>UNIQUEIDENTIFIER</td>
<td>UNIQUEIDENTIFIER</td>
<td></td>
</tr>
<tr>
<td>UNSIGNED BIGINT</td>
<td>UNSIGNED BIGINT</td>
<td></td>
</tr>
<tr>
<td>UNSIGNED INT</td>
<td>UNSIGNED INT</td>
<td></td>
</tr>
<tr>
<td>UNSIGNED SMALLINT</td>
<td>SMALLINT</td>
<td></td>
</tr>
<tr>
<td>UNSIGNED TINYINT</td>
<td>TINYINT</td>
<td></td>
</tr>
<tr>
<td>VARBINARY(n)</td>
<td>VARBINARY(n)</td>
<td></td>
</tr>
<tr>
<td>VARBIT(n)</td>
<td>VARCHAR(n)</td>
<td></td>
</tr>
<tr>
<td>VARCHAR(n)</td>
<td>VARCHAR(n)</td>
<td></td>
</tr>
<tr>
<td>XML</td>
<td>LONG BINARY / IMAGE</td>
<td>There are some restrictions on CHAR and VARCHAR columns over 255 bytes. For more information, see the SAP Sybase IQ documentation.</td>
</tr>
</tbody>
</table>

**Mapping to SQL Anywhere or UltraLite remote data types**

The following table identifies how SAP Sybase IQ 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.

<table>
<thead>
<tr>
<th>SAP Sybase IQ</th>
<th>SQL Anywhere or UltraLite data type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>BIGINT</td>
<td></td>
</tr>
<tr>
<td>BINARY(n)</td>
<td>BINARY(n)</td>
<td></td>
</tr>
<tr>
<td>BIT</td>
<td>BIT</td>
<td></td>
</tr>
<tr>
<td>CHAR(n)</td>
<td>VARCHAR(n)</td>
<td></td>
</tr>
</tbody>
</table>
### MobiLink data mappings between remote and consolidated databases

<table>
<thead>
<tr>
<th>SAP Sybase IQ</th>
<th>SQL Anywhere or UltraLite data type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>DATE</td>
<td></td>
</tr>
<tr>
<td>DATETIME</td>
<td>DATETIME</td>
<td></td>
</tr>
<tr>
<td>DECIMAL((p,s))</td>
<td>DECIMAL((p,s))</td>
<td></td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td></td>
</tr>
<tr>
<td>FLOAT((p))</td>
<td>FLOAT((p))</td>
<td></td>
</tr>
<tr>
<td>INT</td>
<td>INT</td>
<td></td>
</tr>
<tr>
<td>LONG BINARY / IMAGE</td>
<td>LONG BINARY / IMAGE</td>
<td></td>
</tr>
<tr>
<td>LONG VARCHAR / TEXT</td>
<td>LONG VARCHAR / TEXT</td>
<td></td>
</tr>
<tr>
<td>MONEY</td>
<td>MONEY</td>
<td></td>
</tr>
<tr>
<td>NUMERIC((p,s))</td>
<td>NUMERIC((p,s))</td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td>REAL</td>
<td></td>
</tr>
<tr>
<td>SMALLDATETIME</td>
<td>SMALLDATETIME</td>
<td></td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SMALLINT</td>
<td></td>
</tr>
<tr>
<td>SMALLMONEY</td>
<td>SMALLMONEY</td>
<td></td>
</tr>
<tr>
<td>TIME</td>
<td>TIME</td>
<td></td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>TIMESTAMP</td>
<td></td>
</tr>
<tr>
<td>TINYINT</td>
<td>TINYINT</td>
<td></td>
</tr>
<tr>
<td>UNIQUEIDENTIFIER</td>
<td>UNIQUEIDENTIFIER</td>
<td></td>
</tr>
<tr>
<td>UNSIGNED BIGINT</td>
<td>UNSIGNED BIGINT</td>
<td></td>
</tr>
<tr>
<td>UNSIGNED INT</td>
<td>UNSIGNED INT</td>
<td></td>
</tr>
<tr>
<td>VARBINARY((n))</td>
<td>VARBINARY((n))</td>
<td></td>
</tr>
<tr>
<td>VARCHAR((n))</td>
<td>VARCHAR((n))</td>
<td></td>
</tr>
</tbody>
</table>
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.

**Note**
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.

**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.
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.

<table>
<thead>
<tr>
<th>Database</th>
<th>ODBC driver</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL Anywhere 16</td>
<td>SQL Anywhere 161</td>
</tr>
<tr>
<td>Oracle Database 10g or Oracle Database 11g</td>
<td>SQL Anywhere 16 - Oracle1</td>
</tr>
<tr>
<td>Microsoft SQL Server</td>
<td>Microsoft SQL Server ODBC driver2</td>
</tr>
<tr>
<td>Sybase Adaptive Server Enterprise 15.0 or later</td>
<td>Sybase Adaptive Server Enterprise driver2</td>
</tr>
<tr>
<td>IBM DB2 LUW 9.5 for Windows, Linux and Unix</td>
<td>IBM DB2 9.5 CLI driver2</td>
</tr>
<tr>
<td>IBM DB2 LUW 9.7 for Windows, Linux and Unix</td>
<td>IBM DB2 9.7 CLI driver2</td>
</tr>
<tr>
<td>MySQL 5.1</td>
<td>MySQL ODBC driver 5.1</td>
</tr>
</tbody>
</table>


2 Not provided with SQL Anywhere version 16. For installation and configuration instructions, see http://www.sybase.com/detail?id=1011880.

SQL Anywhere 16 - Oracle ODBC driver

The SQL Anywhere 16 - 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.
<table>
<thead>
<tr>
<th>Windows ODBC Data Source Administrator</th>
<th>Configuration for dbdsn command line or <code>.odbc.ini</code> file</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data source name</td>
<td>For dbdsn, use the <code>-w</code> option.</td>
<td>A name to identify your data source.</td>
</tr>
<tr>
<td>User ID</td>
<td>User ID</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>In dbdsn, set this option in the connection string.</td>
<td></td>
</tr>
<tr>
<td>Password</td>
<td>Password</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>In dbdsn, set this option in the connection string.</td>
<td></td>
</tr>
<tr>
<td>TNS service name</td>
<td>Service Name</td>
<td>The TNS Service Name that is stored in <code>network/admin/tnsnames.ora</code> under your Oracle installation directory.</td>
</tr>
<tr>
<td>Encrypt Password</td>
<td>For dbdsn, use the <code>-pe</code> option.</td>
<td>Select this checkbox if you want the password to be stored in encrypted form in the data source.</td>
</tr>
<tr>
<td></td>
<td>Not supported for <code>.odbc.ini</code>.</td>
<td></td>
</tr>
<tr>
<td>Procedure returns results or uses VARRAY parameters</td>
<td>ProcResults</td>
<td>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 <code>download_cursor</code> or <code>download_delete_cursor</code> scripts are stored procedure invocations, select this checkbox.</td>
</tr>
<tr>
<td></td>
<td>In dbdsn, set this option in the connection string.</td>
<td>If no stored procedures use VARRAYs and none of them returns a result set, clear this checkbox to improve performance.</td>
</tr>
<tr>
<td></td>
<td>ProcOwner</td>
<td></td>
</tr>
<tr>
<td></td>
<td>In dbdsn, set this option in the connection string.</td>
<td></td>
</tr>
<tr>
<td>Array Size</td>
<td>ArraySize</td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>In dbdsn, set this option in the connection string.</td>
<td></td>
</tr>
</tbody>
</table>
Creating an ODBC data source for the Oracle driver in Windows

Use this procedure to create an ODBC data source for the Oracle driver in Windows.

**Prerequisites**

There are no prerequisites for this task.

**Create an ODBC data source for the Oracle driver in Windows**

1. Open the ODBC Administrator:
   - Click Start » Programs » SQL Anywhere 16 » Administration Tools » Open ODBC Data Source Administrator.
   
   The **ODBC Data Source Administrator** appears.

2. Click **Add**.

3. Choose **SQL Anywhere 16 - Oracle** and click **Finish**.

4. Specify the configuration options you need. The fields are explained above.

5. Click **Test Connection**, and then click **OK**.

**Results**

The ODBC data source for the Oracle driver is created.

**Next**

Use the ODBC data source to connect.

**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):

<table>
<thead>
<tr>
<th>Windows ODBC Data Source Administrator</th>
<th>Configuration for dbdsn command line or <code>.odbc.ini</code> file</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enable Microsoft distributed transactions</td>
<td>For dbdsn, use the <code>enable-MSDIC</code> option in the connection string. Not supported for <code>.odbc.ini</code>.</td>
<td>Select this checkbox to enlist your transactions in the Microsoft Distributed Transaction Coordinator. When selected, the Oracle ODBC driver requires an Oracle binary file, <code>oramts10.dll</code> for Oracle Database 10g clients or <code>oramts11.dll</code> for Oracle Database 11g clients.</td>
</tr>
</tbody>
</table>
[sample_dsn_using_the_ias_odbc_driver_for_oracle]
Driver=full-path/libdboraodbc12_r.so
UserID=user-id
Password=password
ServiceName=TNS-service-name
ProcResults=[yes|no]
ArraySize=bytes

For an explanation of each field, see the table in “SQL Anywhere 16 - Oracle ODBC driver” on page 714.

**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 in “SQL Anywhere 16 - Oracle ODBC driver” on page 714.

For example:

```
dbdsn -w MyOracleDSN -or -pe -c
"Userid=dba;Password=sql;ServiceName=abcd;ArraySize=100000;ProcResults=y;enableMSDIC=n"
```

See “Data Source utility (dbdsn)” [SQL Anywhere Server - Database Administration].

**See also**

- Recommended ODBC drivers for MobiLink: http://www.sybase.com/detail?id=1011880
- “Data Source utility (dbdsn)” [SQL Anywhere Server - Database Administration]

**MobiLink application 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** can help with your deployment on Windows. See “The Deployment Wizard” [SQL Anywhere Server - Programming].

**Note**

**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.
MobiLink server deployment

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 $SQLANY16/bin32 for 32-bit environments and $SQLANY16/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, SQLANY16, and SQLANYSAMP16.
- 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 %SQLANY16%\Bin32 for 32-bit environments or %SQLANY16%\Bin64 for 64-bit environments. If both entries exist, remove the path that does not apply to your environment.
- To use Java synchronization logic you must have JRE 1.6.0 or later installed, and to use the graphical administration tools (Sybase Central and the MobiLink Profiler), you must have JRE 1.7.0 or later installed.
- To deploy Sybase Central, see “Administration tool deployment” [SQL Anywhere Server - Programming].
- There is a deployment wizard for Windows. See “The Deployment Wizard” [SQL Anywhere Server - Programming].

Windows 32-bit applications

All directories are relative to %SQLANY16%. For more details on the file structure of a 64-bit Windows environment, see “Windows 64-bit applications” on page 721.
<table>
<thead>
<tr>
<th>Description</th>
<th>Windows files</th>
</tr>
</thead>
</table>
| MobiLink server                                 | ● Bin32\mlodbc16.dll  
● Bin32\mlsrv16.exe  
● Bin32\mlsrv16.lic  
● Bin32\mlsql16.dll  
● Bin32\dbicu16.dll  
● Bin32\dbicudt16.dll |
| Language library                                | ● Bin32\dblgen16.dll                                                          |
| Java synchronization logic                      | ● Java\activation.jar  
● Java\imap.jar  
● Java\jodbc4.jar  
● Java\mailapi.jar  
● Java\mlscript.jar  
● Java\mlsupport.jar  
● Java\pop3.jar  
● Java\smtp.jar  
● Bin32\mljava16.dll  
● Bin32\dbjodbc16.dll  
● Bin32\vmjdbc16.dll |
| .NET synchronization logic                      | ● MobiLink\Setup\Dnet\mlDomConfig.xml  
● Bin32\mldnet16.dll  
● Bin32\dnetodbc16.dll  
● Assembly\V2\iAnywhere.MobiLink.dll  
● Assembly\V2\iAnywhere.MobiLink.Script.dll  
● Assembly\V2\iAnywhere.MobiLink.Script.xml  
● Bin32\mlDomConfig.xsd |
| Security option                                 | ● Bin32\mlrsa_tls16.dll  
● Bin32\mlrsa_tls_fips16.dll  
● Bin32\libeay32.dll  
● Bin32\ssleay32.dll  
● Bin32\msvcr90.dll |
| Setup scripts (deploy the ones for your          | ● MobiLink\Setup\  
● MobiLink\Upgrade\ |
<p>| consolidated database)                          | mluser utility                                                               |</p>
<table>
<thead>
<tr>
<th>Description</th>
<th>Windows files</th>
</tr>
</thead>
</table>
| mcreplay utility                                 | • Bin32\mlreplay.exe  
• Bin32\mlgenreplayapi.exe                                                                 |
| MobiLink arbiter                                 | • Bin32\dbserv16.dll  
• Bin32\mlarb16.exe  
• Bin32\mlarb16.lic  
• Bin32\mlarbiter.bat  
• MobiLink\mlarbiter.control  
• mlarbstop.exe                                                                 |
| MobiLink Profiler                                | • Java\mlprof.jar  
• Java\mlstream.jar  
• Java\JComponents1600.jar  
• Java\jsyblib1600.jar  
• Bin32\jsyblib1600.dll  
• Bin32\mlprof.exe  
• Bin32\mljstrm16.dll  
For security with the MobiLink Profiler:  
• Bin32\mlcrsa16.dll  
• Bin32\mlcrsafips16.dll  
• Bin32\mlczlib16.dll                                                                 |
| Online help for the MobiLink 16 plug-in and MobiLink Profiler | • \Documentation\en\htmlhelp\sqlanywhere_en16.chm  
• \Documentation\en\htmlhelp\sqlanywhere_en16.map                                                                 |
| Notifier                                         | • Java\activation.jar  
• Java\jodbc4.jar  
• Java\mailapi.jar  
• Java\mlnotif.jar  
• Java\mlscript.jar  
• Java\smtp.jar  
• Bin32\mljodbc16.dll  
• Bin32\mljstrm16.dll  
For security with the Notifier:  
• Bin32\mlcrsa16.dll  
• Bin32\mlcrsafips16.dll  
• Bin32\mlczlib16.dll                                                                 |
## Description

<table>
<thead>
<tr>
<th>Windows files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bin32\rsoe.exe</td>
</tr>
</tbody>
</table>

For security with the Outbound Enabler:

<table>
<thead>
<tr>
<th>Windows files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bin32\mlcrsa16.dll</td>
</tr>
</tbody>
</table>

## Integrated Outbound Enabler

<table>
<thead>
<tr>
<th>Windows files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bin32\rsosupp16.dll</td>
</tr>
</tbody>
</table>

1 For German, Japanese, and Chinese editions, use dblgde16.dll, dblgja16.dll and dblgzh16.dll respectively.

2 If you are redistributing an application, you must obtain these files directly from Oracle.

3 FIPS requires 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 16 - Introduction].

4 To compile generated code, any files prefixed with mlreplay in SDK/Include are required.

### Windows 64-bit applications

All directories are relative to %SQLANY16%. For more details on the file structure of a 32-bit Windows environment, see “Windows 32-bit applications” on page 718.

<table>
<thead>
<tr>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobiLink server</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Windows files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bin64\mlodbc16.dll</td>
</tr>
<tr>
<td>Bin64\mlsrv16.exe</td>
</tr>
<tr>
<td>Bin64\mlsrv16.lic</td>
</tr>
<tr>
<td>Bin64\mlsql16.dll</td>
</tr>
<tr>
<td>Bin64\dbicu16.dll</td>
</tr>
<tr>
<td>Bin64\dbicudt16.dll</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Language library</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Windows files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bin64\dblgen16.dll</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Java synchronization logic</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Windows files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java\activation.jar</td>
</tr>
<tr>
<td>Java\imap.jar</td>
</tr>
<tr>
<td>Java\jodbc4.jar</td>
</tr>
<tr>
<td>Java\mailapi.jar</td>
</tr>
<tr>
<td>Java\mlscript.jar</td>
</tr>
<tr>
<td>Java\mlsupport.jar</td>
</tr>
<tr>
<td>Java\pop3.jar</td>
</tr>
<tr>
<td>Java\smtp.jar</td>
</tr>
<tr>
<td>Bin64\mljava16.dll</td>
</tr>
<tr>
<td>Bin64\dbjodbc16.dll</td>
</tr>
<tr>
<td>Bin64\mljodbc16.dll</td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>.NET synchronization logic</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Security option</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Setup scripts (deploy the ones for your</td>
</tr>
<tr>
<td>consolidated database)</td>
</tr>
<tr>
<td>mluser utility</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>mlstop utility</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>mlreplay utility</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>MobiLink arbiter</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------------------------------</td>
</tr>
</tbody>
</table>
| MobiLink Profiler                                     | ● Java\mlprof.jar  
● Java\mlstream.jar  
● Java\JComponents1600.jar  
● Java\jsyblib1600.jar  
● Bin64\jsyblib1600.dll  
● Bin64\mlprof.exe  
● Bin64\mljstrm16.dll  
For security with the MobiLink Profiler:³  
● Bin64\mlcrsa16.dll  
● Bin64\mlcrsafips16.dll  
● Bin64\mlczlib16.dll |
| Online help for the MobiLink 16 plug-in and MobiLink Profiler | ● \Documentation\en\htmlhelp\sqlanywhere_en16.chm¹  
● \Documentation\en\htmlhelp\sqlanywhere_en16.map¹ |
| Notifier                                              | ● Java\activation.jar²  
● Java\jodbc4.jar  
● Java\mailapi.jar²  
● Java\mlnotif.jar  
● Java\mlscript.jar  
● Java\smtp.jar²  
● Bin64\mljodbc16.dll  
● Bin64\mljstrm16.dll  
For security with the Notifier:³  
● Bin64\mlcrsa16.dll  
● Bin64\mlcrsafips16.dll  
● Bin64\mlczlib16.dll |
| Relay Server Outbound Enabler                         | ● Bin64\rsoe.exe  
For security with the Outbound Enabler:  
● Bin64\mlcrsa16.dll |
| Integrated Outbound Enabler                           | ● Bin64\rsoesupp16.dll |

¹ For German, Japanese, and Chinese editions, use dblgde16.dll, dblgja16.dll and dblgzh16.dll respectively.

² If you are redistributing an application, you must obtain these files directly from Oracle.
3 FIPS requires 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 16 - Introduction].

4 To compile generated code, any files prefixed with mlreplay in SDK/Include are required.

### Unix 32-bit applications on Unix, Linux, and Mac OS X

All directories are relative to $SQLANY16. For more details on the file structure of a 64-bit Unix environment, see “Unix 64-bit applications on Unix and Linux” on page 726.

<table>
<thead>
<tr>
<th>Description</th>
<th>Unix files</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobiLink server</td>
<td>● bin32/mlsrv16&lt;br&gt;● bin32/mlsrv16.lic&lt;br&gt;● lib32/libdbodm16.so²&lt;br&gt;● lib32/libmlodbc16_r.so²&lt;br&gt;● lib32/libmlsql16_r.so²&lt;br&gt;● lib32/libdbtasks16_r.so²&lt;br&gt;● lib32/libdbicu16_r.so²&lt;br&gt;● lib32/libdbicudt16_r.so²&lt;br&gt;● lib32/libdbodbcinst16_r.so²</td>
</tr>
<tr>
<td>Language library</td>
<td>● res/dblgen16.res²</td>
</tr>
<tr>
<td>Java synchronization logic</td>
<td>● java/activation.jar²&lt;br&gt;● java/imap.jar²&lt;br&gt;● java/jdb4.jar&lt;br&gt;● java/mailapi.jar²&lt;br&gt;● java/mlscript.jar&lt;br&gt;● java/mlsupport.jar&lt;br&gt;● java/pop3.jar²&lt;br&gt;● java/smtp.jar²&lt;br&gt;● lib32/libmljava16_r.so³&lt;br&gt;● lib32/libmljdbc16.so³</td>
</tr>
<tr>
<td>.NET synchronization logic</td>
<td>● Not applicable</td>
</tr>
<tr>
<td>Security option⁴</td>
<td>● lib32/libmlrsa_tls16_r.so³&lt;br&gt;● lib32/libmlrsa_tls_fips16_r.so (on Linux only)&lt;br&gt;● lib32/libcrypto.so (on Linux only)&lt;br&gt;● lib32/libssl.so (on Linux only)</td>
</tr>
<tr>
<td>Setup scripts (deploy the ones for your consolidated database)</td>
<td>● mobilink/setup&lt;br&gt;● mobilink/upgrade</td>
</tr>
<tr>
<td>Description</td>
<td>Unix files</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>
| mluser utility               | • bin32/mluser  
                          • lib32/libmlodbc16_r.so³  
                          • lib32/libdbicu16.so³  
                          • lib32/libdbicudt16.so³ |
| mlstop utility               | • bin32/mlstop  
                          • lib32/libdbicu16.so³ |
| mlreplay utility⁶            | • bin32/mlreplay  
                          • bin32/mlgenreplayapi |
| MobiLink arbiter             | • bin32/libdbserv16_r.so  
                          • bin32/mlarb16  
                          • bin32/mlarb16.lic  
                          • bin32/mlarbiter.sh  
                          • mobilink/mlarbiter.control  
                          • mlarbstop.exe |
| MobiLink Profiler            | • bin32/mlprof  
                          • java/mlprof.jar  
                          • java/mlstream.jar  
                          • java/JComponents1600.jar  
                          • java/jsyblib1600.jar  
                          • lib32/libjsyblib1600_r.so³  
                          For security with the Profiler:  
                          • lib32/libmlcrsa16_r.so  
                          • lib32/libmcrsafips16_r.so  
                          • lib32/libmlczlib16_r.so |
| Notifier                     | • java/activation.jar²  
                          • java/jodbc4.jar  
                          • java/mailapi.jar²  
                          • java/mlnotif.jar  
                          • java/mlscript.jar  
                          • java/smtp.jar²  
                          • lib32/libmljstrm16_r.so³ |
| Relay Server Outbound Enabler| • bin32/rsoe  
                          For security with the Outbound Enabler:  
                          • lib32/libmlcrsa16_r.so³ |
| Integrated Outbound Enabler  | • lib32/librsoesupp16_r.so |
For German, Japanese, and Chinese editions, use dblgde16.dll, dblgja16.dll and dblgzh16.dll respectively.

If you are redistributing an application, you must obtain these files directly from Oracle.

For Linux, the file extension is .so. For Mac OS X, the file extension is .dylib.

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 16 - Introduction].

To compile generated code, any files prefixed with mlreplay in sdk/include are required.

### Unix 64-bit applications on Unix and Linux

All directories are relative to $SQLANY16. 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 724.

<table>
<thead>
<tr>
<th>Description</th>
<th>Unix files</th>
</tr>
</thead>
</table>
| MobiLink server              | ● bin64/mlsrv16  
                                ● bin64/mlsrv16.lic  
                                ● lib64/libdbdm16.so3  
                                ● lib64/libmlodbc16_r.so3  
                                ● lib64/libmlsql16_r.so3  
                                ● lib64/libdbtasks16_r.so3  
                                ● lib64/libdbicu16_r.so3  
                                ● lib64/libdbicudt16_r.so3  
                                ● lib64/libdbodbcinst16_r.so3 |
| Language library             | ● res/dblgen16.res1                                                       |
| Java synchronization logic   | ● java/activation.jar2  
                                ● java/imap.jar2  
                                ● java/jdbc4.jar  
                                ● java/mailapi.jar2  
                                ● java/mlscript.jar  
                                ● java/mlsupport.jar  
                                ● java/pop3.jar2  
                                ● java/smtp.jar2  
                                ● lib64/libmljava16_r.so3  
                                ● lib64/libmljdbc16.so3 |
| .NET synchronization logic   | ● Not applicable                                                        |
| Security option4             | ● lib64/libmlrsa_tls16_r.so3  
                                ● lib64/libmlrsa_tls_fips16_r.so (on Linux only)  
                                ● lib64/libcrypto.so (on Linux only)  
                                ● lib64/libssl.so (on Linux only) |
<table>
<thead>
<tr>
<th>Description</th>
<th>Unix files</th>
</tr>
</thead>
</table>
| Setup scripts (deploy the ones for your consolidated database)             | • mobilink/setup  
• mobilink/upgrade                                                                                                                    |
| mluser utility                                                             | • bin64/mluser  
• lib64/libmlodbc16_r.so<sup>3</sup>  
• lib64/libdbicu16.so<sup>3</sup>  
• lib64/libdbicudt16.so<sup>3</sup>                                                                                         |
| mlstop utility                                                             | • bin64/mlstop  
• lib64/libdbicu16.so<sup>3</sup>                                                                                                        |
| mlreplay utility<sup>6</sup>                                               | • bin64/mlreplay  
• bin64/mlgenraplyapi                                                                                                                    |
| MobiLink arbiter                                                           | • bin64/libdbserv16_r.so  
• bin64/mlarb16  
• bin64/mlarb16.lic  
• bin64/mlarbiter.sh  
• mobilink/mlarbiter.control  
• mlarbstop.exe                                                                                                                        |
| MobiLink Profiler                                                          | • bin64/mlprof  
• java/mlprof.jar  
• java/mlstream.jar  
• java/JComponents1600.jar  
• java/jsylib1600.jar  
• lib64/libjsylib1600_r.so<sup>3</sup>                                                                                             |
|                                                                             | For security with the Profiler:  
• lib64/libmlcrsa16_r.so  
• lib64/libmlcrsafips16_r.so  
• lib64/libmlczlib16_r.so                                                                                                         |
| Notifier                                                                   | • java/activation.jar<sup>2</sup>  
• java/jdbc4.jar  
• java/mailapi.jar<sup>2</sup>  
• java/mlnotif.jar  
• java/mlscript.jar  
• java/smtp.jar<sup>2</sup>                                                                                                         |
| Relay Server Outbound Enabler                                              | • bin64/rsoe                                                                                                                                |
|                                                                             | For security with the Outbound Enabler:  
• lib64/libmlcrsa16_r.so<sup>3</sup>                                                                                                     |
### Description and Unix files

<table>
<thead>
<tr>
<th>Description</th>
<th>Unix files</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrated Outbound Enabler</td>
<td><em>lib64Nibrsoesupp16_r.so</em></td>
</tr>
</tbody>
</table>

1. For German, Japanese, and Chinese editions, use `dblgde16.dll`, `dblgja16.dll` and `dblgzh16.dll` respectively.

2. If you are redistributing an application, you must obtain these files directly from Oracle.

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 16 - Introduction].

5. To compile generated code, any files prefixed with `mlreplay` in `sdk/include` are required.

---

### SQL Anywhere MobiLink client deployment

[This topic has been updated for build 1823.]

### Notes

- For SQL Anywhere clients, you need to deploy a SQL Anywhere database server and the MobiLink client.
  
  See “Database and application deployment” [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 “Administration tool deployment” [SQL Anywhere Server - Programming].

- There is a deployment wizard for Windows. See “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 and .NET assemblies are placed in the `ce\Assembly\v2` directory.

### Windows applications

All directories are relative to `%SQLANY16%`. There are 64-bit versions of these files in the `Bin64` directory.
<table>
<thead>
<tr>
<th>Description</th>
<th>Windows files</th>
</tr>
</thead>
</table>
| MobiLink synchronization client (dbmsync)       | ● Bin32\dbcon16.dll<sup>2</sup>
    | ● Bin32\dbicu16.dll<sup>3</sup>
    | ● Bin32\dblgen16.dll<sup>1</sup>
    | ● Bin32\dblib16.dll
    | ● Bin32\dbmsync.exe
    | ● Bin32\dbtool16.dll<sup>2</sup>                  |
| Dbmsync C++ API or the SQL SYNCHRONIZE statement | MobiLink synchronization client files
    | ● Bin32\dbmsynccli16.dll                          |
| Dbmsync .NET API                                 | MobiLink synchronization client files
    | ● Assembly\V2\iAnywhere.MobiLink.Client.dll      |
| Security option<sup>2</sup>                      | ● Bin32\mlcrsa16.dll
    | ● Bin32\mlcrsafips16.dll
    | ● Bin32\libeay32.dll
    | ● Bin32\ssleay32.dll
    | ● Bin32\msvcr90.dll                               |
| Microsoft ActiveSync utility                     | ● Bin32\mlasinst.exe
    | ● Bin32\mlasdesk.dll
    | ● Bin32\dbcon16.exe
    | ● CE\chip\mlasdev.dll (where chip can be any
    | supported chip, such as arm.50)                  |
| Listener                                         | ● Bin32\dblgen16.dll<sup>1</sup>
    | ● Bin32\dblsn.exe
    | ● Bin32\lsn_udp16.dll
    | ● Bin32\lsn_swi510.dll
    | ● Bin32\maac555.dll
    | ● Bin32\maac750.dll
    | ● Bin32\maac750r3.dll
    | ● Bin32\mabridge.dll
<pre><code>| ● dblsn_sms16.dll&lt;sup&gt;4&lt;/sup&gt;                      |
</code></pre>
<table>
<thead>
<tr>
<th>Description</th>
<th>Windows files</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobiLink Agent (required for central administration of remote databases)</td>
<td>To manage a SQL Anywhere remote database, the following files must be on the remote device:</td>
</tr>
<tr>
<td></td>
<td>Certain features of SQL Anywhere may need extra files to deploy. See “Database and application deployment” [SQL Anywhere Server - Programming].</td>
</tr>
</tbody>
</table>

1 For German, Japanese, and Chinese editions, use dblgde16.dll, dblgja16.dll and dblgzh16.dll 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 $SQLANY16.
MobiLink synchronization client

<table>
<thead>
<tr>
<th>Description</th>
<th>Unix files</th>
</tr>
</thead>
</table>
| MobiLink synchronization client | - bin32/dbmlsync
- res/dblgen16.res
- lib32/libdbicu16_r.so
- lib32/libdblib16_r.so
- lib32/libdbtool16_r.so |
| Dbmsync C++ API or the SQL SYNCHRONIZE statement | - MobiLink synchronization client files
- lib32/libdbmsynccli16_r.so |
| Security option\(^2\) | - lib32/libmlcrsa16_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 16 - Introduction].

UltraLite MobiLink client deployment

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.

<table>
<thead>
<tr>
<th>Description</th>
<th>Windows files</th>
</tr>
</thead>
</table>
| MobiLink Agent | To manage an UltraLite remote database, the following files must be on the remote device:
- dblgen16.dll\(^1\)
- mlagent.exe
- mlastop.exe
- mlauladapt16.dll
- uleng16.exe |

\(^1\) For German, Japanese, and Chinese editions, use dblgde16.dll, dblgja16.dll and dblgzh16.dll respectively.

See also

- “UltraLite deployment” [UltraLite - Database Management and Reference]
- C/C++: “How to build and deploy UltraLite C++ applications” [UltraLite - C and C++ Programming]
- .NET: “Lesson 5: Building and deploying the application” [UltraLite - .NET Programming]
Index

Symbols
-a option
  MobiLink server option (mlsrv16), 46
-ap option
  MobiLink replay utility (mlreplay), 656
-b option
  MobiLink server option (mlsrv16), 46
-bn option
  MobiLink server option (mlsrv16), 47
-c option
  MobiLink Profiler (mlprof), 177
  MobiLink server option (mlsrv16), 48
  MobiLink user authentication utility (mluser), 654
-ca option
  MobiLink server option (mlsrv16), 48
-cinit option
  MobiLink server option (mlsrv16), 49
-classic option
  MobiLink mlsrv16 -sl java option, 72
-classpath option
  MobiLink mlsrv16 -sl java option, 72
-clrConGC option
  MobiLink mlsrv16 -sl dnet option, 71
-clrFlavor option
  MobiLink mlsrv16 -sl dnet option, 71
-clrVersion option
  MobiLink mlsrv16 -sl dnet option, 71
-cm option
  MobiLink server option (mlsrv16), 49
-cmax option
  MobiLink server option (mlsrv16), 50
-cmin option
  MobiLink server option (mlsrv16), 50
-cn option
  MobiLink server option (mlsrv16), 51
-cp option
  MobiLink mlsrv16 -sl java option, 72
-cr option
  MobiLink server option (mlsrv16), 52
-cs option
  MobiLink server option (mlsrv16), 52
-ct option
  MobiLink server option (mlsrv16), 53
-d option
  MobiLink generated replay API utility (mlgenreplayapi), 662
  MobiLink mlsrv16 -sl java option, 72
  MobiLink user authentication utility (mluser), 654
dl option
  MobiLink server option (mlsrv16), 53
  MobiLink user authentication utility (mluser), 654
-DMLStartClasses option
  Java user-defined start classes, 434
  MobiLink mlsrv16 -sl java option, 72
dr option
  MobiLink server option (mlsrv16), 53
ds option
  MobiLink server option (mlsrv16), 54
dsd option
  MobiLink server option (mlsrv16), 54
dt option
  MobiLink server option (mlsrv16), 55
e option
  MobiLink server option (mlsrv16), 55
esu option
  MobiLink server option (mlsrv16), 56
et option
  MobiLink server option (mlsrv16), 57
-f option
  MobiLink replay utility (mlreplay), 656
  MobiLink user authentication utility (mluser), 654
-ftr option
  MobiLink server option (mlsrv16), 58
-fru option
  MobiLink server option (mlsrv16), 58
-h option
  MobiLink stop utility (mlstop), 652
-hotspot option
  MobiLink mlsrv16 -sl java option, 72
-jrepath option
  MobiLink mlsrv16 -sl java option, 72
-lt option
  MobiLink replay utility (mlreplay), 656
-ls option
  MobiLink replay utility (mlreplay), 656
-lsc option
  MobiLink server option (mlsrv16), 59
-MLAutoLoadPath option
  about, 508
  MobiLink mlsrv16 -sl dnet option, 71
-MLDomConfigFile option
  about, 508
MobiLink mlsrv16 -sl dnet option, 71
-MLStartClasses
  .NET user-defined start classes, 503
  MobiLink mlsrv16 -sl dnet option, 71
-n option
  MobiLink replay utility (mlreplay), 656
  MobiLink user authentication utility (mluser), 654
-nc option
  MobiLink server option (mlsrv16), 59
-notifier option
  MobiLink server option (mlsrv16), 60
-o option
  MobiLink generated replay API utility (mlgenreplayapi), 662
  MobiLink replay utility (mlreplay), 656
  MobiLink server option (mlsrv16), 61
  MobiLink user authentication utility (mluser), 654
-on option
  MobiLink server option (mlsrv16), 61
-oq option
  MobiLink server option (mlsrv16), 62
-os option
  MobiLink replay utility (mlreplay), 656
  MobiLink server option (mlsrv16), 62
  MobiLink user authentication utility (mluser), 654
-ot option
  MobiLink generated replay API utility (mlgenreplayapi), 662
  MobiLink replay utility (mlreplay), 656
  MobiLink server option (mlsrv16), 63
  MobiLink user authentication utility (mluser), 654
-p option
  MobiLink Profiler (mlprof), 177
  MobiLink replay utility (mlreplay), 656
  MobiLink user authentication utility (mluser), 654
-pc option
  MobiLink user authentication utility (mluser), 654
-ping option
  MobiLink replay utility (mlreplay), 656
-ppv option
  MobiLink server option (mlsrv16), 63
-q option
  MobiLink server option (mlsrv16), 68
  MobiLink stop utility (mlstop), 652
-r option
  MobiLink Profiler (mlprof), 177
  MobiLink replay utility (mlreplay), 656
  MobiLink server option (mlsrv16), 68
  MobiLink user authentication utility (mluser), 654
-rd option
  MobiLink server option (mlsrv16), 69
-rep option
  MobiLink replay utility (mlreplay), 656
-rg option
  MobiLink replay utility (mlreplay), 656
-rnt option
  MobiLink replay utility (mlreplay), 656
-rp option
  MobiLink replay utility (mlreplay), 656
-rrp option
  MobiLink server option (mlsrv16), 69
-s option
  MobiLink server option (mlsrv16), 71
-sci option
  MobiLink replay utility (mlreplay), 656
-server option
  MobiLink mlsrv16 -sl java option, 72
-sl dnet option
  MobiLink server option (mlsrv16), 71
  user-defined start classes, 503
  using -MLAutoLoadPath, 500
  using -MLDomConfigFile, 508
-sl java option
  MobiLink server option (mlsrv16), 72
  user-defined start classes, 434
-sm option
  MobiLink server option (mlsrv16), 74
-sv option
  MobiLink replay utility (mlreplay), 656
-t option
  MobiLink stop utility (mlstop), 652
-tc option
  MobiLink server option (mlsrv16), 75
-tf option
  MobiLink server option (mlsrv16), 76
-ts option
  MobiLink server option (mlsrv16), 76
-tx option
  MobiLink server option (mlsrv16), 78
-u option
  MobiLink Profiler (mlprof), 177
  MobiLink replay utility (mlreplay), 656
  MobiLink user authentication utility (mluser), 654
-ud option
  MobiLink server option (mlsrv16), 78
-ui option  
  MobiLink server option (mlsrv16), 78
-urc option  
  MobiLink performance benefits, 136
-ux option  
  MobiLink server option (mlsrv16), 79
-v option  
  MobiLink server option (mlsrv16), 79
  MobiLink SQL Anywhere client utility (dbmlsync) performance, 135
-v+ option  
  MobiLink server option (mlsrv16), 79
-vc option  
  MobiLink server option (mlsrv16), 79
-ve option  
  MobiLink server option (mlsrv16), 79
-verbose option  
  MobiLink mlsrv16 -sl java option, 72
-vf option  
  MobiLink server option (mlsrv16), 79
-vh option  
  MobiLink server option (mlsrv16), 79
-vm option  
  MobiLink server option (mlsrv16), 79
-vn option  
  MobiLink server option (mlsrv16), 79
-vp option  
  MobiLink server option (mlsrv16), 79
-vr option  
  MobiLink server option (mlsrv16), 79
-vs option  
  MobiLink server option (mlsrv16), 79
-vt option  
  MobiLink server option (mlsrv16), 79
-vu option  
  MobiLink server option (mlsrv16), 79
-w option  
  MobiLink server option (mlsrv16), 83
  MobiLink stop utility (mlstop), 652
-wm option  
  MobiLink server option (mlsrv16), 84
-wn mlsrv16 option  
  performance, 132
-wn option  
  MobiLink server option (mlsrv16), 75
-wu option  
  MobiLink server option (mlsrv16), 84
-x option

MobiLink mlsrv16 -sl java option, 72
MobiLink Profiler (mlprof), 177
MobiLink replay utility (mlreplay), 656
MobiLink server option (mlsrv16), 85
-zf option  
  MobiLink server option (mlsrv16), 94
-zp option  
  MobiLink server option (mlsrv16), 94
-zs option  
  MobiLink server option (mlsrv16), 95
-shared server state, 95
-zt option  
  MobiLink server option (mlsrv16), 95
-zu option  
  MobiLink server option (mlsrv16), 95
-zus option  
  MobiLink server option (mlsrv16), 96
-zwd option  
  MobiLink server option (mlsrv16), 96
-zwe option  
  MobiLink server option (mlsrv16), 97

.NET  
  MobiLink data types, 501
  MobiLink object-based data flow, 585
  MobiLink synchronization scripts, 497

.NET classes  
  instantiation for .NET synchronization logic, 501

.NET CLR  
  MobiLink options, 71

.NET MobiLink server API (see MobiLink server API for .NET)  

.NET synchronization example  
  MobiLink .NET synchronization logic, 510

.NET synchronization logic  
  .NET class instantiations, 501
deployment, 506
deploying on 32-bit Unix, 724
deploying on 32-bit Windows, 718
deploying on 64-bit Unix, 726
deploying on 64-bit Windows, 721
methods, 502
MobiLink performance, 136
sample, 510
setup, 498
supported languages, 497

.NET synchronization techniques
about, 508
@data option
MobiLink replay utility (mlreplay), 656
MobiLink server option (mlsrv16), 45
MobiLink stop utility (mlstop), 652
MobiLink user authentication utility (mluser), 654
@EmployeeID variable
using with MobiLink primary key pools, 113

A
a.
MobiLink named parameter prefix, 216
MobiLink user-defined parameter prefix, 231
active property
MobiLink Profiler synchronization statistics, 191
Adaptive Server Enterprise
begin_connection_autocommit event, 276
MobiLink consolidated database, 9
MobiLink data mapping, 666
MobiLink isolation levels, 129
MobiLink synchronization, 10
StaticCursorLongColBuffLen, 10
using DDL in MobiLink, 276
Add method
DBParameterCollection class [MobiLink server .NET API], 535
add table script wizard
using, 237
addErrorListener method
ServerContext interface [MobiLink server Java API], 469
addInfoListener method
ServerContext interface [MobiLink server Java API], 469
adding
MobiLink .NET connection scripts, 610
MobiLink .NET table scripts, 611
MobiLink Java connection scripts, 612
MobiLink Java table scripts, 614
MobiLink properties, 621
MobiLink SQL connection scripts, 609
MobiLink SQL table scripts, 624
synchronization scripts with Sybase Central, 236
user names in MobiLink, 653
adding or deleting scripts
MobiLink, 236
adding script versions
MobiLink, 234
adding scripts
MobiLink about, 235
adding synchronization scripts
using stored procedures, 236
addShutdownListener method
ServerContext interface [MobiLink server Java API], 470
addWarningListener method
ServerContext interface [MobiLink server Java API], 471
administration polling interval
MobiLink agent, 153
AdventureWorks
synchronization issues, 15
agent database
definition, 143
agent ID
about, 142
agent_db_file
remote tasks, 172
agent_id
remote tasks, 172
agent_log_file
remote tasks, 172
all way synchronization (see bi-directional synchronization)
antialiasing
MobiLink Profiler option, 184
APIs
MobiLink server API for Java, 440
application servers
synchronizing with MobiLink, 585
application-to-application messaging
(see also) messaging
applications
deploying MobiLink, 717
array size
Oracle driver option, 715
ASE
(see also) Adaptive Server Enterprise
assemblies
implementing in MobiLink, 508
locating in MobiLink .NET synchronization logic, 498
authenticate_file_transfer
connection event, 260
authenticate_file_upload
authenticate_user
connection event, 266
authenticate_user property
authenticate_user_hashed
connection event, 271
authentication
MobiLink user authentication utility (mluser), 653
authentication parameters
MobiLink, 231
MobiLink scripts, 216
authentication_status synchronization parameter
about, 266
AUTOINCREMENT
(see also GLOBAL AUTOINCREMENT)
autoincrement methods
Oracle MobiLink consolidated databases, 19
automatic validation
MobiLink file-based download, 200
AvantGo (see UltraLite for M-Business Anywhere)

B
battery_level
remote tasks, 172
begin_connection
connection event, 275
begin_connection_autocommit
connection event, 276
begin_download
connection event, 277
table event, 278
begin_download_deletes
table event, 281
begin_download_rows
table event, 282
begin_publication
connection event, 283
begin_sync property
MobiLink Profiler synchronization statistics, 191
begin_synchronization
connection event, 287
table event, 289
begin_upload
connection event, 292
table event, 294
begin_upload_deletes
table event, 296
begin_upload_rows
table event, 299
bi-directional synchronization
about, 107
required scripts, 235
BLOBs
downloaded from ASE, 10
blocking download acknowledgement
about, 126
bottlenecks
MobiLink performance, 136
broadcast download
MobiLink file-based download, 195
buffer_size option
MobiLink mlsrv16 -x option for OE, 91
buffer_size protocol option
MobiLink mlsrv16 -x option for HTTP, 88
MobiLink mlsrv16 -x option for HTTPS, 89

C
C# programming language
MobiLink .NET support, 497
MobiLink options, 71
MobiLink synchronization scripts, 497
C++ programming language
MobiLink .NET support, 497
central administration of remote databases
about, 140
adding recipient to a deployed task, 164
agent database definition, 143
canceling a deployed task, 162
command definition, 142
commands, 166
concepts, 141
copy file command, 166
create database command, 166
creating a remote task, 159
delete file command, 167
deployed task definition, 142
deploying a remote task, 160
download file command, 167
drop database command, 168
editing a remote task, 160
execute SQL command, 168
exporting a remote task, 161
initiating a remote task, 163
MobiLink agent definition, 142
MobiLink agent id definition, 142
MobiLink project definition, 141
prompt command, 169
reactivating a remote task, 164
remote database definition, 143
remote schema name definition, 143
remote task definition, 142
remote tasks, 155
rename command, 155
run program command, 170
server-initiated remote task definition, 143
setup, 143
status, 173
status information definition, 143
stopping the agent, 150
Sybase Central, 151
synchronize command, 170
system procedures, 174
upload file command, 170
working with deployed tasks, 161
central administration of remote tasks (see central administration of remote databases) SIRT, 165
central databases
MobiLink consolidated databases, 1
changing the last download time
MobiLink, 102
CHAR columns
ASE MobiLink consolidated databases, 10
IBM DB2 MobiLink consolidated databases, 12
MobiLink issues, 5
MobiLink server option (mlsrv16) -b, 46
Oracle MobiLink consolidated databases, 19
SQL Server MobiLink consolidated databases, 14
CHAR data type
MobiLink and other DBMSs, 5
character set considerations
MobiLink, 712
character set conversion
by ODBC drivers, 713
during MobiLink synchronization, 712
character sets
MobiLink synchronization, 712
chart pane
MobiLink Profiler, 184
class instances
Java synchronization logic, 429
MobiLink .NET synchronization logic, 501
CLASSPATH environment variable
MobiLink Java synchronization logic, 427
Clear method
DBParameterCollection class [MobiLink server .NET API], 536
client event-hook procedures
(see also event hooks)
ClientCertificates property
NetworkData interface [MobiLink server .NET API], 558
Close method
DBCommand interface [MobiLink server .NET API], 521
DBConnection interface [MobiLink server .NET API], 523
DBRowReader interface [MobiLink server .NET API], 544
CLR
MobiLink options, 71
collation sequences
MobiLink synchronization, 712
collect_network_data protocol option
MobiLink mlsrv16 -x option for HTTP, 88
MobiLink mlsrv16 -x option for HTTPS, 89
MobiLink mlsrv16 -x option for TCP/IP, 86
MobiLink mlsrv16 -x option for TLS over TCP/IP, 87
collisions
MobiLink conflict resolution, 115
column sizes
ASE MobiLink consolidated databases, 10
ColumnNames property
DBRowReader interface [MobiLink server .NET API], 544
ColumnTypes property
DBRowReader interface [MobiLink server .NET API], 545
command line
MobiLink server (mlsrv16), 40
starting mlsrv16, 40
command line utilities
MobiLink generated replay API (mlgenreplayapi) syntax, 662
MobiLink replay (mlreplay) syntax, 656
MobiLink stop (mlstop) syntax, 652
Contains method
DBParameterCollection class [MobiLink server .NET API], 536

contention
MobiLink performance, 131
MobiLink performance explanation, 137

conversion
character set by ODBC drivers, 713
conversion between character sets
MobiLink synchronization, 712

CopyTo method
DBParameterCollection class [MobiLink server .NET API], 537

Count property
DBParameterCollection class [MobiLink server .NET API], 541

create connection script wizard
using, 236

create script version wizard
using, 234

create service wizard
MobiLink, 32

create your java synchronization script
MobiLink Java synchronization logic example, 437

CreateAndInitMLReplayUploadTransaction callback
[mlreplaycallbacks.cpp]
description, 597

CreateCommand method
DBConnection interface [MobiLink server .NET API], 524

CreateSpatialValue method
SpatialUtilities class [MobiLink server .NET API], 568

createSpatialValue method
SpatialUtilities class [MobiLink server Java API], 479

creating
download file for MobiLink file-based download, 197
file-definition database, 196
MobiLink consolidated databases, 2

creating consolidated databases
MobiLink about, 2
creating databases
consolidated, 2
creating download files
MobiLink file-based download, 197
creating file-definition databases

MobiLink, 196
cursor scripts
defined, 215
custom validation
MobiLink file-based download, 202
customizing your statistics
MobiLink Profiler, 188

D
d daemon
running MobiLink as a, 31
data consistency
(see also synchronization)
data exchange (see synchronization)
data flow (MobiLink) (see direct row handling)
data inconsistency
MobiLink conflict-handling, 115
data mappings
about, 666
data scripts
about, 258
download_cursor, 301
download_delete_cursor, 303
handle_DownloadData, 341
handle_UploadData, 352
upload_delete, 401
upload_fetch, 403
upload_fetch_column_conflict, 405
upload_insert, 407
upload_new_row_insert, 409
upload_old_row_insert, 411
upload_update, 422
data source name
Oracle driver option, 715
data type mapping
MobiLink consolidated databases, 666
data types
MobiLink .NET and SQL, 501
MobiLink consolidated database mappings, 666
MobiLink IBM DB2 LUW, 675
MobiLink Java and SQL, 430
MobiLink mapping ASE, 666
MobiLink mapping SAP HANA, 704
MobiLink mapping SAP Sybase IQ, 708
MobiLink Microsoft SQL Server, 683
MobiLink MySQL, 690
MobiLink Oracle, 695
database connections
  MobiLink performance, 139
  MobiLink performance, setting maximum, 134
database schemas
  relating consolidated tables to MobiLink remote
  tables, 2
database tools interface
  (see also DBTools interface)
database worker threads
  automatic adjustment, 133
  MobiLink, 137
  MobiLink performance, 133
databases
  MobiLink consolidated databases, 1
DateTime property
  DateTimeWithTimeZone class [MobiLink
  server .NET API], 517
DateTimeWithTimeZone class [MobiLink
  server .NET API]
  DateTime property, 517
  DateTimeWithTimeZone constructor, 513
  Day property, 518
  description, 512
  Hour property, 518
  Millisecond property, 518
  Minute property, 518
  Month property, 519
  Parse method, 516
  Second property, 519
  TimeZoneHour property, 519
  TimeZoneMinute property, 519
  ToString method, 516
  Year property, 520
DateTimeWithTimeZone constructor
  DateTimeWithTimeZone class [MobiLink
  server .NET API], 513
Day property
  DateTimeWithTimeZone class [MobiLink
  server .NET API], 518
daylight savings time
  MobiLink, 102
db_location
  remote tasks, 172
DBCommand interface [MobiLink server .NET API]
  Close method, 521
  CommandText property, 522
  description, 520
  ExecuteNonQuery method, 521
  Parameters property, 522
  Prepare method, 522
DBConnection interface [MobiLink server .NET API]
  Close method, 523
  Commit method, 524
  CreateCommand method, 524
  description, 523
  Rollback method, 524
DBConnectionContext
  constructors, 502
DBConnectionContext interface [MobiLink server .NET API]
  description, 524
  GetConnection method, 526
  GetDownloadData method, 526
  GetProperties method, 527
  GetRemoteID method, 527
  GetServerContext method, 528
  GetVersion method, 529
  NetworkData property, 529
DBConnectionContext interface [MobiLink server Java API]
  description, 440
  getConnection method, 441
  getDownloadData method, 442
  getNetworkData method, 443
  getProperties method, 443
  getRemoteID method, 444
  getServerContext method, 444
  getVersion method, 445
dblsn utility
  MobiLink client deployment on Windows, 728
dbmlsync (see dbmlsync utility)
dbmlsync utility
  deploying, 728
  deploying on Unix, 730
  deploying on Windows, 728
DBParameter class [MobiLink server .NET API]
  DbType property, 531
  description, 530
  Direction property, 531
  HasChanged field, 533
  IsNullable property, 532
  ParameterName property, 532
  Precision property, 532
  Scale property, 533
  Size property, 533
Value property, 533

DBParameterCollection class [MobiLink server .NET API]
   Add method, 535
   Clear method, 536
   Contains method, 536
   CopyTo method, 537
   Count property, 541
   DBParameterCollection constructor, 535
   description, 534
   GetEnumerator method, 537
   IndexOf method, 538
   Insert method, 539
   IsFixedSize property, 541
   IsReadOnly property, 541
   IsSynchronized property, 541
   Remove method, 539
   RemoveAt method, 540
   SyncRoot property, 541
   this property, 542
   DBParameterCollection constructor
      DBParameterCollection class [MobiLink server .NET API], 535

DBRowReader interface [MobiLink server .NET API]
   Close method, 544
   ColumnNames property, 544
   ColumnTypes property, 545
   description, 543
   NextRow method, 544

DBTools interface
   (see also database tools interface)

DbType property
   DBParameter class [MobiLink server .NET API], 531
   debugging
      .NET synchronization logic, 506
      MobiLink connections, 39
      MobiLink server log, 29
      MobiLink synchronization using Java classes, 432
debbuging .NET synchronization logic
      about, 506
debbuging Java classes
      MobiLink Java synchronization logic, 432
default global autoincrement
   MobiLink declaring, 110
default isolation levels
   MobiLink, 129

DelayCreationOfSimulatedClient callback
   [mlreplaycallbacks.cpp]
   description, 598

DelayDestructionOfSimulatedClient callback
   [mlreplaycallbacks.cpp]
   description, 598

DelayStartOfReplay callback [mlreplaycallbacks.cpp]
   description, 599
deletes
   MobiLink downloads, 244
   stopping upload of for SQL Anywhere clients, 123
deleting
   MobiLink .NET connection scripts, 610
   MobiLink .NET table scripts, 611
   MobiLink Java connection scripts, 612
   MobiLink Java table scripts, 614
   MobiLink properties, 621
   MobiLink SQL connection scripts, 609
   MobiLink SQL table scripts, 624
   rows in remote MobiLink databases, 244
deleting all the rows in a table
   MobiLink, 245
deleting rows
   MobiLink remote databases, 244
   MobiLink techniques, 122
deleting rows with the download_delete_cursor script
   MobiLink, 244
deleting script version (see removing script version)
deleting scripts
   MobiLink about, 235
deployed remote task
   adding recipients, 164
   canceling, 162
   definition, 142
   initiating, 163
   reactivating, 164
   working with, 161
deployed task (see deployed remote task)
deploying
   MobiLink applications, 717
   MobiLink applications and databases, 717
   MobiLink performance, 131
   MobiLink server, 718
   overview of MobiLink, 717
   SQL Anywhere MobiLink clients, 728
   UltraLite MobiLink clients, 731
deploying MobiLink applications
   about, 717
deploying remote databases  
about, 717

deploying SQL Anywhere MobiLink clients  
about, 728

deploying the MobiLink server  
about, 718

deploying UltraLite MobiLink clients  
about, 731

deployment (see deploying)  
deployment overview  
MobiLink, 717

DestroyMLReplayUploadTransaction callback  
[mlreplaycallbacks.cpp]  
description, 599
details table pane  
MobiLink Profiler, 182
detecting conflicts  
MobiLink, 116
MobiLink with upload_fetch scripts, 116
MobiLink with upload_update scripts, 118
development tips  
MobiLink direct row handling, 587
MobiLink synchronization, 98
direct inserts of scripts  
MobiLink, 238
direct row access (see direct row handling)  
direct row handling  
about, 585
development tips, 587
downloads, 594
handle_DownloadData connection event, 341
handle_UploadData connection event, 352
quick start, 586
uploads, 588
direct synchronization events  
about, 586
Direction property  
DBParameter class [MobiLink server .NET API], 531
disjoint partitioning  
defined, 104
MobiLink, 104
distributable download  
MobiLink file-based download, 195
domain configuration files  
MobiLink, 509
download acknowledgement  
about, 126
download events  
MobiLink synchronization, 257
download failure  
MobiLink restartable downloads, 124
download file  
creating for MobiLink file-based download, 197
download only  
(see also download-only)
download only synchronization  
(see also download-only synchronization)
download property  
MobiLink Profiler synchronization statistics, 191
download timestamp  
about MobiLink, 100
MobiLink generation of, 101
download transaction  
MobiLink, 251
download-only synchronization  
about, 107
required scripts, 235
download_bytes property  
MobiLink Profiler synchronization statistics, 191
download_cursor  
about, 243
example using a stored procedure call, 127
MobiLink disjoint partitioning, 104
partitioning foreign key tables, 107
partitioning with overlaps, 106
performance, 135
table event, 301
timestamp-based synchronization, 100
using a stored procedure call, 127
writing scripts to download rows, 242
download_delete_cursor  
about, 244
disjoint partitioning, 104
example using a stored procedure call, 127
partitioning foreign key tables, 107
partitioning with overlaps, 106
performance, 135
table event, 303
using a stored procedure call, 127
writing scripts to download rows, 242
download_deleted_rows property  
MobiLink Profiler synchronization statistics, 191
download_errors property  
MobiLink Profiler synchronization statistics, 191
download_fetched_rows property
MobiLink Profiler synchronization statistics, 191
  download_filtered_rows property
  MobiLink Profiler synchronization statistics, 191
download_statistics
  connection event, 305
table event, 308
download_timestamp
  MobiLink generation of, 101
download_warnings property
  MobiLink Profiler synchronization statistics, 191
DownloadData interface [MobiLink server .NET API]
  description, 545
  GetDownloadTableByName method, 546
  GetDownloadTables method, 546
DownloadData interface [MobiLink server Java API]
  description, 445
  getDownloadTableByName method, 446
  getDownloadTables method, 447
downloading a result set from a stored procedure call
  synchronization techniques, 127
downloading data
  file-based download in MobiLink, 195
downloading deletes
  MobiLink download_delete_cursor scripts, 244
downloading rows
  synchronization scripts, 242
downloads
  file-based MobiLink, 195
  MobiLink failed downloads, 124
  MobiLink performance, 136
  MobiLink scripts to download rows, 242
  MobiLink transaction, 251
timestamp-based, 99
DownloadTableData interface [MobiLink server .NET API]
  description, 547
  GetDeleteCommand method, 549
  GetLastDownloadTime method, 549
  GetName method, 550
  GetSchemaTable method, 550
  GetUpsertCommand method, 550
DownloadTableData interface [MobiLink server Java API]
  description, 448
  getDeletePreparedStatement method, 450
  getUpsertPreparedStatement method, 454
drivers
  supported by MobiLink, 714
duration property
  MobiLink Profiler synchronization statistics, 191
dynamic cache sizing
  MobiLink server, 37

E

e2ee_private_key protocol option
  MobiLink mlsrv16 -x option for HTTPS, 89
  MobiLink mlsrv16 -x option for TCP/IP, 87
e2ee_private_key_password protocol option
  MobiLink mlsrv16 -x option for HTTPS, 89
  MobiLink mlsrv16 -x option for TCP/IP, 87
empty strings
  Oracle MobiLink consolidated databases, 19
  Oracle not supported, 20
enabling Microsoft distributed transactions
  Oracle driver option, 715
encrypting passwords
  Oracle driver option, 715
end-to-end encryption type
  MobiLink server option (mlsrv16), 85
download
  connection event, 312
  connection event, 313
table event, 315
download_deletes
  table event, 318
download_rows
  table event, 319
download_warnings
  MobiLink Profiler synchronization statistics, 191
download
  connection event, 321
download
  connection event, 329
table event, 331
download_deletes
  table event, 334
download_rows
  table event, 336
ending
  MobiLink server, 29
enterprise databases
  synchronizing with MobiLink, 585
equals method
  TimestampWithTimeZone class [MobiLink server Java API], 484
error handling
  during MobiLink synchronization, 246
error logs
  MobiLink server option (mlsrv16), 55
ERROR variable
  LogMessage class [MobiLink server Java API], 460
ErrorListener event
  ServerContext interface [MobiLink server .NET API], 565
errors
  handling during MobiLink synchronization, 246
  MobiLink modify_error_message connection event, 358
  recording, 248
event model
  MobiLink pseudocode, 252
events
  about MobiLink events, 212
  about MobiLink synchronization, 248
  MobiLink, 248
  MobiLink direct row handling, 586
events during download
  about, 257
  writing scripts to download rows, 242
events during upload
  about, 255
  writing scripts to upload rows, 239
Excel
  synchronizing with MobiLink, 585
ExecuteNonQuery method
  DBCommand interface [MobiLink server .NET API], 521
ExecuteReader method
  DBCommand interface [MobiLink server .NET API], 522

failed downloads
  MobiLink, 124

synchronization techniques, 124
failover
  MobiLink server farm, 36
farm (see MobiLink server farm)
_FATALException class [MobiLink server .NET API]
  description, 551
_FATALException constructor, 551
_FATALException constructor
  __FATALException class [MobiLink server .NET API], 551
file-based downloads
  about, 195
  examples, 202
file-definition database
  about, 196
  creating, 196
_file_authentication_code
  authenticate_file_transfer parameter, 260
files
  MobiLink file-based download, 195
_FiniIdentifySimulatedClient callback
  [mlreplaycallbacks.cpp]
  description, 600
_FIPS option
  MobiLink server option (mlsrv16), 57
  MobiLink user authentication utility (mluser), 654
_FIPS protocol option
  MobiLink mlsrv16 -x option for HTTPS, 89
  MobiLink mlsrv16 -x option using TCP/IP, 87
fragmentation
  (see also partitioning)
FTP
  MobiLink file-based download, 195
full synchronization (see bi-directional synchronization)
fundamental rules
  MobiLink, 98

G

generate_next_last_download_timestamp
  connection event, 339
generation numbers
  MobiLink file-based download, 201
GetBytes method
  SpatialUtilities class [MobiLink server .NET API], 569
getBytes method
SpatialUtilities class [MobiLink server Java API], 480
getCertificateChain method
NetworkData interface [MobiLink server Java API], 465
GetConnection method
DBConnectionContext interface [MobiLink server.NET API], 526
getConnection method
DBConnectionContext interface [MobiLink server Java API], 441
GetDeleteCommand method
DownloadTableData interface [MobiLink server .NET API], 549
getDeletePreparedStatement method
DownloadTableData interface [MobiLink server Java API], 450
GetDeletes method
UploadedTableData interface [MobiLink server .NET API], 576
getDeletes method
UploadedTableData interface [MobiLink server Java API], 494
GetDownloadApplyTime callback
[mlreplaycallbacks.cpp]
description, 601
GetDownloadData method
DBConnectionContext interface [MobiLink server .NET API], 526
getDownloadData method
DBConnectionContext interface [MobiLink server Java API], 441
GetDownloadTableByName method
DownloadData interface [MobiLink server .NET API], 546
getDownloadTableByName method
DownloadData interface [MobiLink server Java API], 446
GetDownloadTables method
DownloadData interface [MobiLink server .NET API], 546
getDownloadTables method
DownloadData interface [MobiLink server Java API], 447
GetGetEnumerator method
DBParameterCollection class [MobiLink server .NET API], 537
getHTTPHeaders method
NetworkData interface [MobiLink server Java API], 466
GetHTTPHeaderValue method
NetworkData interface [MobiLink server .NET API], 557
getHTTPHeaderValue method
NetworkData interface [MobiLink server Java API], 466
GetHTTPHeaderValues method
NetworkData interface [MobiLink server .NET API], 557
getHTTPHeaderValues method
NetworkData interface [MobiLink server Java API], 466
GetInserts method
UploadedTableData interface [MobiLink server .NET API], 577
getInserts method
UploadedTableData interface [MobiLink server Java API], 494
GetLastDownloadTime method
DownloadTableData interface [MobiLink server .NET API], 549
getLastDownloadTime method
DownloadTableData interface [MobiLink server Java API], 451
getMetaData method
DownloadData interface [MobiLink server .NET API], 546
getMetaData method
DownloadData interface [MobiLink server Java API], 446
GetMLReplayAPIVersion callback
[mlreplaycallbacks.cpp]
description, 602
GetName method
DownloadData interface [MobiLink server .NET API], 550
GetName method
DownloadData interface [MobiLink server Java API], 453
getMLReplayAPIVersion method
DownloadData interface [MobiLink server .NET API], 578
getMLReplayAPIVersion method
DownloadData interface [MobiLink server Java API], 496
getNetworkData method
DBConnectionContext interface [MobiLink server Java API], 443
GetProperties method
DBConnectionContext interface [MobiLink server .NET API], 529
getVersion method
DBConnectionContext interface [MobiLink server Java API], 445
global
script versions in MobiLink, 233
global assembly cache
implementing in MobiLink, 508
GLOBAL AUTOINCREMENT
(see also AUTOINCREMENT)
global autoincrement
algorithm, 111
MobiLink declaring, 110
MobiLink unique primary keys, 109
setting global_database_id for MobiLink, 111
global script versions
MobiLink, 233
global_database_id option
setting in MobiLink, 111
GlobalFini callback [mlreplaycallbacks.cpp]
description, 603
GlobalInit callback [mlreplaycallbacks.cpp]
description, 603
graph pane
MobiLink Profiler, 183
groups
MobiLink, 154
GUIDs
(see also UUIDs)
H
handle_DownloadData
connection event, 341
handle_error
connection event, 344
synchronization scripts, 246
handle_odbc_error
connection event, 349
handle_UploadData
connection event, 352
handling conflicts
MobiLink, 115
MobiLink direct row handling, 589
handling conflicts for direct uploads
MobiLink direct row handling, 589
handling deletes
MobiLink, 122
handling direct downloads
MobiLink direct row handling, 594
Handling direct uploads
MobiLink direct row handling, 588
handling errors
MobiLink server, 344
handling MobiLink server errors in Java
MobiLink Java synchronization logic, 433
handling MobiLink server errors with .NET
MobiLink .NET synchronization logic, 505
handling multiple errors in a single SQL statement
MobiLink, 246
hard shutdown
MobiLink stop utility (mlstop), 652
HasChanged field
DBParameter class [MobiLink server .NET API], 533
header_limit protocol option
MobiLink mlsrv16 -x option for HTTP, 88
MobiLink mlsrv16 -x option for HTTPS, 89
MobiLink mlsrv16 -x option for OE, 91
hooks
(see also event hooks)
host protocol option
MobiLink mlsrv16 -x option for HTTP, 88
MobiLink mlsrv16 -x option for HTTPS, 89
MobiLink mlsrv16 -x option for TCP/IP, 86
MobiLink mlsrv16 -x option for TLS over TCP/IP, 87
hosted Relay Server
(see also Sybase hosted relay service)
Hour property
DateTimeWithTimeZone class [MobiLink server .NET API], 518
how conflicts are detected
MobiLink, 116
how default values are chosen
MobiLink global autoincrement, 111
HTTP
MobiLink mlsrv16 -x option, 88
HTTPHeaders property
NetworkData interface [MobiLink server .NET API], 558
HTTPS
MobiLink mlsrv16 -x option, 89
DBParameterCollection class [MobiLink server .NET API], 541
isTLS method
  NetworkData interface [MobiLink server Java API], 467
IsTLS property
  NetworkData interface [MobiLink server .NET API], 559

J
Java
  MobiLink data types, 430
  MobiLink object-based data flow, 585
  MobiLink server API reference, 440
  MobiLink synchronization scripts, 427
Java classes
  instantiation for Java synchronization logic, 429
Java MobiLink server API (see MobiLink server API for Java)
Java synchronization
  MobiLink Java synchronization logic, 436
Java synchronization logic
  deploying on 32-bit Unix, 724
  deploying on 32-bit Windows, 718
  deploying on 64-bit Unix, 726
  deploying on 64-bit Windows, 721
Java class instantiations, 429
methods, 431
  MobiLink performance, 136
  MobiLink server API, 440
sample, 436
setup, 427
  specifying in MobiLink server command line, 427
Java VM
  MobiLink options, 72
Java vs. SQL synchronization logic
  MobiLink performance, 136
Javadoc
  MobiLink, 440

K
keep partial download synchronization parameter
  restartable downloads, 125
key pools
  MobiLink synchronization application, 112
killing
  MobiLink server, 29

L
language libraries
  MobiLink server deployment on 32-bit Unix, 724
  MobiLink server deployment on 32-bit Windows, 718
  MobiLink server deployment on 64-bit Unix, 726
  MobiLink server deployment on 64-bit Windows, 721
last download time
  about MobiLink, 100
last download timestamp
  about MobiLink, 100
  handle_DownloadData connection event, 339
  MobiLink generation of, 101
  modify_last_download_timestamp connection event, 360
  modify_next_last_download_timestamp connection event, 363
last modified column
  MobiLink, 99
last_download
  MobiLink named parameter, 100
  modify_last_download_timestamp connection event, 360
last_download_timestamp
  MobiLink generation of, 101
  MobiLink named parameter, 100
last_table_download
  MobiLink named parameter, 100
  modify_last_download_timestamp connection event, 360
Linux
  deploying dbmlsync, 730
Listener utility (dblsn)
  MobiLink client deployment on Windows, 728
load balancing
  MobiLink server farm, 36
loading assemblies
  MobiLink .NET synchronization logic, 508
lock escalation
  IBM DB2 MobiLink consolidated databases, 12
log file viewer
  MobiLink client deployment on Windows, 728
MobiLink mlsrv16 -x option for HTTP, 88
MobiLink mlsrv16 -x option for HTTPS, 89
MobiLink mlsrv16 -x option for OE, 91
LogCallback delegate [MobiLink server .NET API]
description, 581
logging
MobiLink performance, 135
MobiLink server actions, 29
logging MobiLink server actions
about, 29
logical deletes
writing download_delete_cursor scripts, 244
LogListener interface [MobiLink server Java API]
description, 458
messageLogged method, 458
LogMessage class [MobiLink server .NET API]
description, 553
LogMessage constructor, 553
MessageType enumeration, 554
Text property, 554
Type property, 554
User property, 555
LogMessage class [MobiLink server Java API]
description, 459
ERROR variable, 460
getText method, 459
crUser method, 459
INFO variable, 461
WARNING variable, 462
LogMessage constructor
LogMessage class [MobiLink server .NET API], 553
logs
(see also log files)
LONG data type
Oracle synchronization, 701
LUW
IBM DB2 LUW as MobiLink consolidated
database, 11

M
Mac OS X
deploying dbmlsync, 730
maintaining unique primary keys
about, 108
composite keys, 108
global autoincrement, 109
primary key pools, 112
UUIDs, 109
MakeConnection method
ServerContext interface [MobiLink server .NET
API], 564
makeConnection method
ServerContext interface [MobiLink server Java
API], 473
Manage Anywhere
MobiLink file-based download, 195
managed remote database
adding, 154
many-to-many relationships
partitioning, 106
synchronization, 106
mapping
MobiLink consolidated database data types, 666
marquee tool
MobiLink Profiler overview pane, 186
memory usage
MobiLink server, 37
messageLogged method
LogListener interface [MobiLink server Java API], 458
MessageType enumeration
LogMessage class [MobiLink server .NET API], 554
methods
MobiLink .NET synchronization logic, 502
MobiLink Java synchronization logic, 431
Microsoft ActiveSync
MobiLink client deployment on Windows, 728
Microsoft Distributed Transaction Coordinator
Oracle driver option, 715
Microsoft Excel
synchronizing with MobiLink, 585
Microsoft SQL Server
as MobiLink consolidated database, 13
MobiLink data mapping, 683
MobiLink isolation levels, 129
Microsoft SQL Server consolidated database
MobiLink, 13
Millisecond property
DateTimeWithTimeZone class [MobiLink
server .NET API], 518
Minute property

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0 751
DateTimeWithTimeZone class [MobiLink server .NET API], 518
ml_add_certificates_file system procedure syntax, 608
ml_add_column system procedure syntax, 608
ml_add_connection_script system procedure syntax, 609
ml_add_dnet_connection_script system procedure syntax, 610
ml_add_dnet_table_script system procedure syntax, 611
ml_add_java_connection_script system procedure syntax, 612
ml_add_java_table_script system procedure syntax, 614
ml_add_lang_connection_script system procedure syntax, 615
ml_add_lang_connection_script_chk system procedure syntax, 615
ml_add_lang_table_script system procedure syntax, 615
ml_add_lang_table_script_chk system procedure syntax, 615
ml_add_lcs system procedure syntax, 615
ml_add_lcs_chk system procedure syntax, 615
ml_add_ldap_server system procedure syntax, 615
ml_add_lts system procedure syntax, 615
ml_add_lts_chk system procedure syntax, 615
ml_add_missing_dnld_scripts system procedure syntax, 617
ml_add_passsthrough system procedure syntax, 617
ml_add_passsthrough_repair system procedure syntax, 618
ml_add_passsthrough_script system procedure syntax, 619
ml_add_property system procedure syntax, 621
ml_add_table_script system procedure syntax, 624
ml_add_user system procedure

ml_add_user_auth_policy system procedure syntax, 626
ml_delete passthrough system procedure syntax, 627
ml_delete_passthrough_repair system procedure syntax, 628
ml_delete_passthrough_script system procedure syntax, 628
ml_delete_sync_state system procedure syntax, 629
ml_delete_sync_state_before system procedure syntax, 630
ml_delete_user system procedure syntax, 631
ml_global script version about, 233
ml_model_check_all_schema system procedure syntax, 632
ml_model_check_version_schema system procedure syntax, 633
ml_model_drop system procedure syntax, 631
ml_password remote tasks, 172
ml_ra_add_agent_id system procedure syntax, 634
ml_ra_assign_task system procedure syntax, 635
ml_ra_cancel_notification system procedure syntax, 636
ml_ra_cancel_task_instance system procedure syntax, 636
ml_ra_clone_agent_properties system procedure syntax, 637
ml_ra_delete_agent_id system procedure syntax, 637
ml_ra_delete_events_before system procedure syntax, 638
ml_ra_delete_remote_id system procedure syntax, 638
ml_ra_delete_task system procedure syntax, 639
ml_ra_get_agent_events system procedure syntax, 639
ml_ra_get_agent_ids system procedure syntax, 641
ml_ra_get_agent_properties system procedure
syntax, 642
ml_ra_get_latest_event_id system procedure
syntax, 643
ml_ra_get_orphan_taskdbs system procedure
syntax, 643
ml_ra_get_remote_ids system procedure
syntax, 644
ml_ra_get_task_results system procedure
syntax, 644
ml_ra_get_task_status system procedure
syntax, 646
ml_ra_manage_remote_db system procedure
syntax, 648
ml_ra_notify_agent_sync system procedure
syntax, 648
ml_ra_notify_task system procedure
syntax, 649
ml_ra_reassign_taskdb system procedure
syntax, 649
ml_ra_set_agent_property system procedure
syntax, 649
ml_ra_unmanage_remote_db system procedure
syntax, 650
ml_reset_sync_state system procedure
syntax, 651
ml_server_delete system procedure
syntax, 651
ml_server_update system procedure
syntax, 651
ml_stream
remote tasks, 172
ml_user
MobiLink user authentication utility (mluser), 653
ml_username
remote tasks, 172
mlagent
syntax, 146
mlagent command
about, 145
running, 145
mlarbiter utility
Unix, 664
Windows, 663
mlarbstop
MobiLink arbiter stop utility, 665
mlastop command
about, 150
mlDomConfig.xml

mlgenreplayapi utility
options, 662
syntax, 662
mlprof
about MobiLink Profiler, 176
options, 177
starting, 177
mlreplay utility
options, 656
syntax, 656
mlreplaycallbacks.cpp reference
CreateAndInitMLReplayUploadTransaction callback, 597
DelayCreationOfSimulatedClient callback, 598
DelayDestructionOfSimulatedClient callback, 598
DelayStartOfReplay callback, 599
DestroyMLReplayUploadTransaction callback, 599
FinIdentifySimulatedClient callback, 600
GetDownloadApplyTime callback, 601
GetMLReplayAPIVersion callback, 602
GetUploadTransaction callback, 602
GlobalFini callback, 603
GlobalInit callback, 603
IdentifySimulatedClient callback, 604
MobiLink Replay C++ callbacks, 597
ReportEndOfReplay callback, 605
mlscript.jar
MobiLink Java synchronization logic, 427
mlsrv16
(see also MobiLink server)
about, 40
alphabetical list of options, 40
connection string, 48
logging, 29
Notifier, 60
options, 40
report error context in message log, 61
starting, 27
stopping, 29
syntax, 40
mlstop utility
deploying on 32-bit Unix, 724
deploying on 32-bit Windows, 718
deploying on 64-bit Unix, 726
deploying on 64-bit Windows, 721
methods for stopping MobiLink server, 29
options, 652
syntax, 652
mluser utility
deploying on 32-bit Unix, 724
deploying on 32-bit Windows, 718
deploying on 64-bit Unix, 726
deploying on 64-bit Windows, 721
options, 653
syntax, 653
Mobile Device Center (see Microsoft ActiveSync)
MobiLink
.NET synchronization logic, 497
alphabetic list of events, 248
character set considerations, 712
connection parameters for mlsrv16, 85
connection parameters for Profiler, 177
consolidated databases, 1
data types, 666
deploying applications, 717
deploying UltraLite clients, 731
development tips, 98
event overview, 248
file-based download, 195
handling conflicts, 115
Java synchronization logic, 427
mlsrv16 options, 40
ODBC driver support, 714
Profiler, 176
running outside the current session, 31
running the synchronization server, 27
scripts, 211
starting, 27
stopping the MobiLink server, 29
synchronization techniques, 98
system procedures, 605
system tables, 4
MobiLink agent
about, 144
adding remote databases, 154
authentication, 155
configuration window, 148
configuring, 145
definition, 142
deploying on Windows for SQL Anywhere clients, 728
deploying on Windows for UltraLite clients, 731
properties, 153
running, 145
stopping, 150
Sybase Central, 151
MobiLink agent ID
definition, 142
MobiLink agent properties
connection strings, 153
MobiLink client network protocol options, 153
polling interval, 153
synchronization interval, 153
viewing and changing, 153
MobiLink arbiter server utility
Unix, 664
Windows, 663
MobiLink clients
deploying, 728
MobiLink connections
debugging, 39
MobiLink connectivity
verifying with HTTP and HTTPS, 28, 39
MobiLink consolidated databases
about, 1
ASE, 9
IBM DB2 LUW as, 11
MySQL as, 15
Oracle as, 18
SAP HANA, 23
SAP Sybase IQ as, 25
SQL Anywhere as, 25
SQL Server as, 13
MobiLink data mappings
about, 666
MobiLink data mappings between remote and consolidated databases
about, 666
MobiLink data types
.NET and SQL, 501
Java and SQL, 430
MobiLink direct conflict handling
.NET example, 592
Java example, 591
MobiLink events
listed, 248
MobiLink farm (see MobiLink server farm)
MobiLink file transfer utility (mlfiletransfer)
MobiLink mlsrv16 -ftr option, 58
MobiLink mlsrv16 -ftru option, 58
MobiLink generated replay API utility
(mlgenreplayapi)
syntax, 662
MobiLink generation numbers
file-based download, 201
MobiLink groups
adding, 154
MobiLink Listener utility (dblsn)
MobiLink client deployment on Windows, 728
MobiLink log file viewer
MobiLink server logs, 30
MobiLink object-based data flow for Java and .NET
about, 585
MobiLink performance
about, 131
key factors, 136
monitoring, 140
MobiLink Profiler
about, 176
chart pane, 184
database, 180
database views, 189
deleting previous session, 180
deploying on 32-bit Unix, 724
deploying on 32-bit Windows, 718
deploying on 64-bit Unix, 726
deploying on 64-bit Windows, 721
details table pane, 182
graph pane, 183
graph pane, using, 184
marquee tool, 185
new watch, 188
opening previous session, 180
options, 177, 186
overview pane, 185
sample properties, 187
session properties, 187
specifying watches, 188
starting, 177, 178
statistical properties, 191
stopping, 179
user interface, 181
using, 181
Watch Manager, 188
zooming, 184
MobiLink project
definition, 141
MobiLink Replay C++ callbacks
mlreplaycallbacks.cpp reference, 597
MobiLink replay utility (mlreplay)
syntax, 656
MobiLink scripts
listed, 248
MobiLink server
(see also mlsrv16)
deploying, 718
MobiLink generated replay API utility
(mlgenreplayapi), 662
MobiLink replay utility (mlreplay), 656
MobiLink stop utility (mlstop), 652
options, 40
running, 27
starting, 27
syntax, 40
MobiLink server .NET API
DateTimeWithTimeZone class, 512
DBCommand interface, 520
DBConnection interface, 523
DBConnectionContext interface, 524
DBParameter class, 530
DBParameterCollection class, 534
DBRowReader interface, 543
DownloadData interface, 545
DownloadTableData interface, 547
FatalException class, 551
LogCallback delegate, 581
LogMessage class, 553
NetworkData interface, 555
ScriptExecutionException class, 559
ServerContext interface, 561
ServerException class, 566
ShutdownCallback delegate, 581
SpatialUtilities class, 568
SQLType enumeration, 581
SynchronizationException class, 570
UpdateDataReader interface, 572
UploadDataReader interface, 573
UploadedTableData interface, 575
MobiLink server .NET API reference
iAnywhere.MobiLink.Script namespace, 512
MobiLink server farm
failover, 36
load balancing, 36
MobiLink mlsrv16 -lsc option, 59
MobiLink server Java API
DBCConnectionContext interface, 440
DownloadData interface, 445
DownloadTableData interface, 448
InOutInteger interface, 455
InOutString interface, 456
LogListener interface, 458
LogMessage class, 459
NetworkData interface, 463
ServerContext interface, 467
ServerException class, 476
ShutdownListener interface, 477
SpatialUtilities class, 479
SynchronizationException class, 480
TimestampWithTimeZone class, 482
UpdateResultSet interface, 487
UploadData interface, 489
UploadedTableData interface, 491
MobiLink server Java API reference
ianywhere.ml.script package, 440
MobiLink server log file viewer
MobiLink server options
about, 40
MobiLink server shared state
server, 36
MobiLink statistical properties
MobiLink Profiler, 191
MobiLink stop utility (mstop)
syntax, 652
MobiLink stored procedures (see MobiLink system procedures)
MobiLink synchronization
.NET synchronization logic, 497
consolidated databases, 1
file-based download, 195
Java synchronization logic, 427
overview of events, 248
performance, 131
restartable downloads, 124
writing .NET classes, 502
writing Java classes, 431
MobiLink synchronization logic
.NET, 497
alphabetic list of scripts, 248
data types for .NET and SQL, 501
data types for Java and SQL, 430
Java, 427
synchronization techniques, 98
writing scripts, 211
MobiLink synchronization scripts
about, 211
alphabetic list of scripts, 248
constructing .NET classes, 502
constructing Java classes, 431
database transactions and .NET classes, 501
database transactions and Java classes, 430
debugging Java classes, 432
preserving database transactions in .NET, 501
preserving database transactions in Java, 430
writing .NET classes, 502
writing Java classes, 431
MobiLink synchronization server (see MobiLink server)
MobiLink system database
about, 3
MobiLink System Database (MLSD)
-Cs option for mlsrv16, 52
MobiLink system procedures
about, 605
MobiLink system tables
about, 4
creating in consolidated database, 2
MobiLink user authentication utility (mluser)
syntax, 653
MobiLink users
MobiLink user authentication utility (mluser), 653
registering with the MobiLink user authentication utility (mluser), 653
MobiLink utilities
about, 652
MobiLink arbiter stop syntax, Windows, 665
MobiLink arbiter syntax, Windows, 663
MobiLink arbiter utility syntax, Unix, 664
MobiLink generated replay API (mlgenreplayapi)
syntax, 662
MobiLink replay (mlreplay) syntax, 656
MobiLink stop (mstop) syntax, 652
MobiLink user authentication (mluser) syntax, 653
server, 652
mobility (see MobiLink)
modify_error_message
collection event, 358
modify_last_download_timestamp
collection event, 360
modify_next_last_download_timestamp
collection event, 363
modify_user
connection event, 366
monitoring
  MobiLink performance, 140
  synchronizations in MobiLink, 176
monitoring MobiLink performance
overview, 140
Month property
  DateTimeWithTimeZone class [MobiLink
  server .NET API], 519
multi-threaded network processing
  MobiLink performance, 132
  MobiLink server option (mlsrv16), 75
MySQL
  MobiLink consolidated database, 15
  MobiLink data mapping, 690
MySQL consolidated database
  MobiLink, 15

N
named parameters
  about MobiLink, 216
  last_download, 100
  last_table_download, 100
  spatial data synchronization, 6
named row parameters
  about MobiLink scripts, 216
  adding column information to the consolidated
database, 608
named script parameters
  about MobiLink, 216
  ml_add_column system procedure, 608
network parameters
  MobiLink mlsrv16 -x option, 85
network protocols
  MobiLink mlsrv16 -x option using TCP/IP, 86
  MobiLink mlsrv16 -x option using TLS over
  TCP/IP, 87
  MobiLink mlsrv16 using HTTP, 88
  MobiLink mlsrv16 using HTTPS, 89
  MobiLink server, 86
NetworkData interface [MobiLink server .NET API]
  ClientCertificates property, 558
description, 555
  GetHTTPHeader method, 557
  GetHTTPHeaderValues method, 557
  HttpHeaders property, 558
  IsEndToEndEncrypted property, 559
  IsHTTP property, 559
  IsTLS property, 559
  NetworkData interface [MobiLink server Java API]
description, 463
  getCertificateChain method, 465
  getHTTPHeaders method, 466
  getHTTPHeaderValues method, 466
  isEndToEndEncrypted method, 467
  isHTTP method, 467
  isTLS method, 467
  NetworkData property
  DBConnectionContext interface [MobiLink
  server .NET API], 529
  NextRow method
  DBRowReader interface [MobiLink server .NET
  API], 544
non-blocking download acknowledgement
  about, 126
  nonblocking_download_ack connection event, 369
  publication_nonblocking_download_ack
  connection event, 375
  non-relational databases
  synchronizing with MobiLink, 585
  nonblocking_download_ack
  connection event, 369
Notifiers
  deploying on 32-bit Unix, 724
  deploying on 32-bit Windows, 718
  deploying on 64-bit Unix, 726
  deploying on 64-bit Windows, 721
  starting, 60
O
  o.
  MobiLink named parameter prefix, 216
object-based data flow (see direct row handling)
objects
  MobiLink server API for Java, 440
ODBC
  MobiLink drivers, 714
  multiple errors in MobiLink, 246
  Oracle driver, 714
ODBC drivers
  MobiLink character set conversion by, 713
  MobiLink support, 714
  Oracle, 714
OE
  MobiLink mlsrv16 -x option, 91
old row parameters
  MobiLink scripts, 216
options
  mlsrv16, 40
  MobiLink generated replay API utility
    (mlgenreplayapi), 662
  MobiLink replay utility (mlreplay), 656
  MobiLink server (mlsrv16), 40
  MobiLink stop utility (mlstop), 652
  MobiLink user authentication utility (mluser), 654
options window
  MobiLink Profiler, 186
Oracle
  as MobiLink consolidated database, 18
  MobiLink data mapping, 695
  MobiLink isolation levels, 129
  ODBC driver, 714
  sequences in MobiLink synchronization, 20
  synchronizing LONG data, 701
  synchronizing spatial data, 7
  XMLTYPE data type, 20
Oracle consolidated database
  MobiLink, 18
Oracle driver
  encrypting passwords, 715
  ODBC, 714
Oracle varray
  example, 22
  restrictions, 22
  using in stored procedures, 22
overlaps
  partitioning, 104
overview pane
  MobiLink Profiler, 185

P
packaged download
  MobiLink file-based download, 195
ParameterName property
  DBParameter class [MobiLink server .NET API], 532
parameters
  synchronization scripts, 216
Parameters property
  DBCommand interface [MobiLink server .NET API], 522
Parse method
  DateTimeWithTimeZone class [MobiLink server .NET API], 516
partial download retained synchronization parameter
  restartable downloads, 125
partitioning
  about MobiLink, 104
  defined, 104
  MobiLink disjoint, 104
partitioning foreign key tables
  MobiLink, 107
partitioning rows
  MobiLink among remote databases, 104
partitioning tables
  example, 104
partitioning with overlaps
  MobiLink, 106
passthrough mode
  (see also SQL passthrough)
passwords
  encrypting in Oracle driver, 715
  MobiLink user authentication utility (mluser), 653
performance
  automatic adjustment of database worker threads, 133
  MobiLink, 131
  MobiLink concurrency, 131
  MobiLink contention, 131
  MobiLink database connections, 139
  MobiLink database worker threads, 133
  MobiLink downloads, 136
  MobiLink incremental uploads, 133
  MobiLink logging verbosity, 135
  MobiLink maximum database connections, 134
  MobiLink memory, 134
  MobiLink mlsrv16 -sm option, 74
  MobiLink row uploads, 136
  MobiLink script execution, 135
  MobiLink synchronization logic, 136
  MobiLink synchronization priority, 136
  MobiLink threading, 133
  MobiLink transactional uploads, 133
  MobiLink tuning, 137
  MobiLink, synchronizing BLOBs, 134
polling interval
  MobiLink agent, 153
port protocol option
  MobiLink mlsrv16 -x option for HTTP, 88
  MobiLink mlsrv16 -x option for HTTPS, 89
  MobiLink mlsrv16 -x option for TCP/IP, 86
  MobiLink mlsrv16 -x option for TLS over TCP/IP, 87
Precision property
  DBParameter class [MobiLink server .NET API], 532
prefixes
  MobiLink named parameters, 216
Prepare method
  DBCCommand interface [MobiLink server .NET API], 522
prepare_for_download
  connection event, 372
prepare_for_download property
  MobiLink Profiler synchronization statistics, 191
primary key pools
  generating unique values using default global
  autoincrement for MobiLink, 109
  MobiLink example, 115
  MobiLink unique primary keys, 112
primary keys
  MobiLink about, 98
  MobiLink uniqueness techniques, 108
  Oracle sequences, 20
printing information from .NET
  MobiLink .NET synchronization logic, 505
priority synchronization
  MobiLink performance, 136
private assemblies
  implementing in MobiLink, 508
privileges
  MobiLink server, 28
procedure calls
  SQL Server MobiLink consolidated databases, 14
  procedures
    MobiLink, 605
  procedures return results
    Oracle driver option, 715
ProcResults
  Oracle driver option, 715
Profiler
  MobiLink Profiler, 176
profiling
  synchronizations in MobiLink, 176
profiling database
  MobiLink, 180
viewing, 189
profiling session
  creating a watch, 188
  deleting previous session, 180
  ending, 179
  opening previous session, 180
  starting, 178
projects
  adding groups, 154
protocols
  (see also network protocols)
  MobiLink mlsrv16 -x option using TCP/IP, 86
  MobiLink mlsrv16 -x option using TLS over
  TCP/IP, 87
  MobiLink mlsrv16 using HTTP, 88
  MobiLink mlsrv16 using HTTPS, 89
  MobiLink server, 86
pseudocode
  MobiLink events, 248
  publication_nonblocking_download_ack
    connection event, 375
Q
question marks
  MobiLink script parameters, 216
quick start
  MobiLink direct row handling, 586
quitting
  MobiLink server, 29
quotation marks
  IBM DB2 MobiLink consolidated databases, 12
R
r.
  MobiLink named parameter prefix, 216
READPAST table hint
  download_cursor problems with, 302
  download_delete_cursor problems with, 305
  upload_fetch problems with, 404
recipients
  adding to a deployed remote task, 164
reconciling data (see synchronization)
registering
  methods as MobiLink scripts, 606
  registering methods
    MobiLink server API for Java, 431
registering MobiLink users
  MobiLink user authentication utility (mluser), 653
Relay Server hosting service
  (see also Sybase hosted relay service)
Relay Server Outbound Enabler
  (see also Outbound Enabler)
relayserver (see Relay Server)
remote administration (see central administration of remote databases)
remote databases
  MobiLink mapping of data types, 666
  relating consolidated tables to MobiLink remote tables, 2
remote schema name
  adding, 151
  definition, 143
  importing, 151
remote tables
  deleting rows in MobiLink, 244
remote task
  about, 155
  adding commands, 171
  commands, 166
  copy file command, 166
  create database command, 166
  creating, 159
  definition, 142
  delete file command, 167
  deploying, 160
  download file command, 167
  drop database command, 168
  execute SQL command, 168
  exporting, 161
  logic, 158
  prompt command, 169
  rename file command, 169
  run program command, 170
  status, 173
  synchronize command, 170
  upload file command, 170
  variables, 172
remote task attribute
  commands, 156
  conditions, 156
  delay between retries, 157
  description, 156
  high priority, 157
  maximum number of retries, 157
maximum running time, 157
name, 156
random delay interval, 157
remote schema name, 156
schema change, 157
status reporting, 157
trigger mechanisms, 156
remote tasks
  editing, 160
  remote_id, 172
Remove method
  DBParameterCollection class [MobiLink server .NET API], 539
RemoveAt method
  DBParameterCollection class [MobiLink server .NET API], 540
removeErrorListener method
  ServerContext interface [MobiLink server Java API], 474
removeInfoListener method
  ServerContext interface [MobiLink server Java API], 474
removeShutdownListener method
  ServerContext interface [MobiLink server Java API], 475
removeWarningListener method
  ServerContext interface [MobiLink server Java API], 475
removing script versions
  MobiLink, 234
replication
  (see also MobiLink)
report_error
  connection event, 378
  syntax, 248
report_odbc_error
  connection event, 381
ReportEndOfReplay callback [mlreplaycallbacks.cpp]
  description, 605
reporting errors
  MobiLink synchronization, 248
required scripts
  MobiLink, 235
resetting
  MobiLink last download time, 102
resolution
  MobiLink conflict resolution, 115
resolve_conflict
  table event, 385
  using, 118
resolving
  MobiLink conflicts, 115
resolving conflicts
  MobiLink overview, 118
  MobiLink with resolve_conflict scripts, 118
  MobiLink with upload_update scripts, 120
  resolve_conflict script, 118
  upload_update script, 120
restartable downloads
  MobiLink, 124
resume partial download synchronization parameter
  restartable downloads, 125
resuming failed downloads
  MobiLink, 124
return values
  .NET synchronization, 503
  Java synchronization, 432
Rollback method
  DBConnection interface [MobiLink server .NET API], 524
row handling in MobiLink (see direct row handling)
row parameters
  MobiLink scripts, 216
row-level versioning
  with MobiLink, 26
rows
  deleting on remote MobiLink databases, 244
  partitioning, 104
rsa protocol option
  MobiLink mlsrv16 -x option for HTTPS, 89
  MobiLink mlsrv16 -x option for TCP/IP, 87
rhost (see Relay Server State Manager)
rsoe (see Outbound Enabler)
rtnotifier
  syntax, 165
rules
  (see also transmission rules)
running
  MobiLink server, 27
running .NET synchronization logic
  about, 498
running Java synchronization logic
  about, 427
running MobiLink
  about, 27
  as a daemon, 31
  outside the current session, 31
running MobiLink server
  as a service, 31
running the MobiLink server
  about, 27
  in a server farm, 36
S
s.
  MobiLink named parameter prefix, 216
sample domain configuration file
  MobiLink, 509
sample properties
  MobiLink Profiler, 187
samples
  .NET synchronization logic, 510
  Java synchronization logic, 436
SAP HANA
  MobiLink consolidated database, 23
  MobiLink data mapping, 704
  MobiLink synchronization, 23
SAP Sybase IQ
  as MobiLink consolidated database, 25
  MobiLink data mapping, 708
SAP Sybase IQ consolidated database
  MobiLink, 25
Scale property
  DBParameter class [MobiLink server .NET API], 533
schemas
  relating consolidated tables to MobiLink remote tables, 2
script parameters
  about MobiLink, 216
  last_download, 100
  last_table_download, 100
script types
  MobiLink, 215
script versions
  about MobiLink synchronization, 232
  adding, 234
  global, 233
  removing, 234
  reserved names, 233
ScriptExecutionException class [MobiLink server .NET API]
### Index

<table>
<thead>
<tr>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ScriptExecutionException constructor</td>
<td>560</td>
</tr>
<tr>
<td>ScriptExecutionException constructor</td>
<td>560</td>
</tr>
<tr>
<td>ScriptExecutionException class [MobiLink server .NET API]</td>
<td>560</td>
</tr>
<tr>
<td>scripts</td>
<td></td>
</tr>
<tr>
<td>about MobiLink</td>
<td>211</td>
</tr>
<tr>
<td>adding and deleting .NET connection scripts</td>
<td>610</td>
</tr>
<tr>
<td>adding and deleting .NET table scripts</td>
<td>611</td>
</tr>
<tr>
<td>adding and deleting Java connection scripts</td>
<td>612</td>
</tr>
<tr>
<td>adding and deleting Java table scripts</td>
<td>614</td>
</tr>
<tr>
<td>adding and deleting SQL connection scripts</td>
<td>609</td>
</tr>
<tr>
<td>adding and deleting SQL table scripts</td>
<td>624</td>
</tr>
<tr>
<td>adding to the consolidated database in MobiLink</td>
<td>235</td>
</tr>
<tr>
<td>connection scripts</td>
<td>215</td>
</tr>
<tr>
<td>global script versions</td>
<td>233</td>
</tr>
<tr>
<td>MobiLink event overview</td>
<td>248</td>
</tr>
<tr>
<td>MobiLink events</td>
<td>248</td>
</tr>
<tr>
<td>required by MobiLink</td>
<td>235</td>
</tr>
<tr>
<td>supported DBMS scripting strategies</td>
<td>4</td>
</tr>
<tr>
<td>synchronizing spatial data</td>
<td>8</td>
</tr>
<tr>
<td>table scripts</td>
<td>215</td>
</tr>
<tr>
<td>versions</td>
<td>232</td>
</tr>
<tr>
<td>writing scripts to download rows</td>
<td>242</td>
</tr>
<tr>
<td>writing scripts to upload rows</td>
<td>239</td>
</tr>
<tr>
<td>scripts for synchronizing spatial data</td>
<td></td>
</tr>
<tr>
<td>IBM DB2 sample</td>
<td>9</td>
</tr>
<tr>
<td>Microsoft SQL Server sample</td>
<td>8</td>
</tr>
<tr>
<td>MySQL sample</td>
<td>9</td>
</tr>
<tr>
<td>Oracle sample</td>
<td>9</td>
</tr>
<tr>
<td>SQL Anywhere sample</td>
<td>8</td>
</tr>
<tr>
<td>upload and download scripts</td>
<td>8</td>
</tr>
<tr>
<td>Second property</td>
<td></td>
</tr>
<tr>
<td>DateTimeWithTimeZone class [MobiLink server .NET API]</td>
<td>519</td>
</tr>
<tr>
<td>security</td>
<td></td>
</tr>
<tr>
<td>MobiLink user authentication utility (mluser)</td>
<td>653</td>
</tr>
<tr>
<td>selective sharing (see partitioning)</td>
<td></td>
</tr>
<tr>
<td>self-referencing tables</td>
<td></td>
</tr>
<tr>
<td>MobiLink</td>
<td>129</td>
</tr>
<tr>
<td>sequence of MobiLink events</td>
<td></td>
</tr>
<tr>
<td>pseudocode</td>
<td>252</td>
</tr>
<tr>
<td>sequences</td>
<td></td>
</tr>
<tr>
<td>primary key uniqueness in MobiLink synchronization</td>
<td>20</td>
</tr>
<tr>
<td>server farm</td>
<td></td>
</tr>
<tr>
<td>load balancing</td>
<td>36</td>
</tr>
<tr>
<td>MobiLink</td>
<td>36</td>
</tr>
<tr>
<td>MobiLink mlsrv16 -zs option</td>
<td>95</td>
</tr>
<tr>
<td>server initiated synchronization (see server-initiated synchronization)</td>
<td></td>
</tr>
<tr>
<td>server system procedures</td>
<td></td>
</tr>
<tr>
<td>MobiLink</td>
<td>605</td>
</tr>
<tr>
<td>server-initiated remote task</td>
<td></td>
</tr>
<tr>
<td>definition</td>
<td>143</td>
</tr>
<tr>
<td>SIRT</td>
<td>165</td>
</tr>
<tr>
<td>ServerContext interface [MobiLink server .NET API]</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>561</td>
</tr>
<tr>
<td>ErrorListener event</td>
<td>565</td>
</tr>
<tr>
<td>getProperties method</td>
<td>562</td>
</tr>
<tr>
<td>getPropertiesByVersion method</td>
<td>563</td>
</tr>
<tr>
<td>getPropertySetNames method</td>
<td>563</td>
</tr>
<tr>
<td>GetStartClassInstances method</td>
<td>564</td>
</tr>
<tr>
<td>InfoListener event</td>
<td>565</td>
</tr>
<tr>
<td>MakeConnection method</td>
<td>564</td>
</tr>
<tr>
<td>Shutdown method</td>
<td>565</td>
</tr>
<tr>
<td>ShutdownListener event</td>
<td>565</td>
</tr>
<tr>
<td>WarningListener event</td>
<td>566</td>
</tr>
<tr>
<td>ServerContext interface [MobiLink server Java API]</td>
<td></td>
</tr>
<tr>
<td>addErrorListener method</td>
<td>469</td>
</tr>
<tr>
<td>addInfoListener method</td>
<td>469</td>
</tr>
<tr>
<td>addShutdownListener method</td>
<td>470</td>
</tr>
<tr>
<td>addWarningListener method</td>
<td>471</td>
</tr>
<tr>
<td>description</td>
<td>467</td>
</tr>
<tr>
<td>getProperties method</td>
<td>471</td>
</tr>
<tr>
<td>getPropertiesByVersion method</td>
<td>472</td>
</tr>
<tr>
<td>getPropertySetNames method</td>
<td>472</td>
</tr>
<tr>
<td>getStartClassInstances method</td>
<td>473</td>
</tr>
<tr>
<td>makeConnection method</td>
<td>473</td>
</tr>
<tr>
<td>removeErrorListener method</td>
<td>474</td>
</tr>
<tr>
<td>removeInfoListener method</td>
<td>474</td>
</tr>
<tr>
<td>removeShutdownListener method</td>
<td>475</td>
</tr>
<tr>
<td>removeWarningListener method</td>
<td>475</td>
</tr>
<tr>
<td>shutdown method</td>
<td>475</td>
</tr>
<tr>
<td>ServerException class [MobiLink server .NET API]</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>566</td>
</tr>
<tr>
<td>ServerException constructor</td>
<td>566</td>
</tr>
<tr>
<td>ServerException class [MobiLink server Java API]</td>
<td></td>
</tr>
<tr>
<td>description</td>
<td>476</td>
</tr>
<tr>
<td>ServerException constructor</td>
<td>477</td>
</tr>
<tr>
<td>ServerException constructor</td>
<td></td>
</tr>
<tr>
<td>ServerException class [MobiLink server .NET API]</td>
<td></td>
</tr>
<tr>
<td>566</td>
<td></td>
</tr>
<tr>
<td>ServerException class [MobiLink server Java API]</td>
<td>477</td>
</tr>
</tbody>
</table>
servers
   MobiLink server utility (mlsrv16), 27
service dependencies
   MobiLink, 35
ServiceName
   Oracle driver option, 715
services
   adding in MobiLink, 32
   configuring, 32
   deleting, 32
   MobiLink, 32
   MobiLink dependencies, 35
   MobiLink running multiple, 35
   MobiLink server, 31
   running MobiLink as a service, 31
session properties
   MobiLink Profiler, 187
session-wide variables
   IBM DB2 MobiLink consolidated databases, 12
   Oracle MobiLink consolidated databases, 19
SET NOCOUNT
   SQL Server MobiLink consolidated databases, 14
SetNewRowValues method
   UpdateDataReader interface [MobiLink
   server .NET API], 573
setNewRowValues method
   UpdateResultSet interface [MobiLink server
   Java API], 488
SetOldRowValues method
   UpdateDataReader interface [MobiLink
   server .NET API], 573
setOldRowValues method
   UpdateResultSet interface [MobiLink server
   Java API], 489
SetSRID method
   SpatialUtilities class [MobiLink server .NET API],
   569
setSRID method
   SpatialUtilities class [MobiLink server Java API],
   480
setTimeZoneOffsetHours method
   TimestampWithTimeZone class [MobiLink server
   Java API], 486
setTimeZoneOffsetMinutes method
   TimestampWithTimeZone class [MobiLink server
   Java API], 486
setting up
   MobiLink consolidated databases, 1
MobiLink file-based downloads, 196
MobiLink Java synchronization logic, 427
setting up .NET synchronization logic
   about, 498
setting up a MySQL consolidated database
   MobiLink, 15
setting up a SQL Anywhere consolidated database
   MobiLink, 25
setting up a Sybase ASE consolidated database
   MobiLink, 9
setting up an IBM DB2 LUW consolidated database
   MobiLink, 11
setting up an Oracle consolidated database
   MobiLink, 18
setting up an SAP HANA consolidated database
   MobiLink, 23
setting up an SAP Sybase IQ consolidated database
   MobiLink, 25
setting up direct row handling
   about, 586
setup
   MobiLink .NET synchronization logic, 498
   MobiLink consolidated databases, 2
   MobiLink Java synchronization logic, 427
setup scripts
   MobiLink consolidated databases, 2
   MobiLink system database, 3
setValue method
   InOutInteger interface [MobiLink server Java
   API], 456
   InOutString interface [MobiLink server Java API],
   458
shadow tables
   writing download_delete_cursor scripts, 244
shared assemblies
   implementing in MobiLink, 508
shared server state
   MobiLink mlsrv16 -zs option, 95
shared stated
   MobiLink server farm, 36
sharing rules (see partitioning)
Shutdown method
   ServerContext interface [MobiLink server .NET
   API], 565
shutdown method
   ServerContext interface [MobiLink server Java
   API], 475
ShutdownCallback delegate [MobiLink server .NET API]
   description, 581
ShutdownListener event
   ServerContext interface [MobiLink server .NET API], 565
ShutdownListener interface [MobiLink server Java API]
   description, 477
   shutdownPerformed method, 478
shutdownPerformed method
   ShutdownListener interface [MobiLink server Java API], 478
shutting down
   MobiLink server, 29
   MobiLink stop utility (mlstop), 652
SIRT (see server-initiated remote task)
   initiating a deployed remote task, 163
   server-initiated remote task, 165
sis (see server-initiated synchronization)
Size property
   DBParameter class [MobiLink server .NET API], 533
snapshot isolation
   MobiLink, 129
   MobiLink mlsrv16 -dsd option to disable, 54
   MobiLink mlsrv16 -dt option for SQL Server, 55
   MobiLink mlsrv16 -esu option to enable for uploads, 56
snapshot synchronization
   about, 102
soft shutdown
   MobiLink stop utility (mlstop), 652
sort order
   characters and MobiLink synchronization, 712
spatial data
   synchronization, 6
spatial data synchronization
   named parameters, 6
   Oracle considerations, 7
   restrictions, 6
   SRID requirements, 6
   upload and download scripts, 8
SpatialUtilities class [MobiLink server .NET API]
   CreateSpatialValue method, 568
   description, 568
   GetBytes method, 569
   GetSRID method, 569
   SetSRID method, 569
SpatialUtilities class [MobiLink server Java API]
   createSpatialValue method, 479
   description, 479
   getBytes method, 480
   getSRID method, 480
   setSRID method, 480
spreadsheets
   synchronizing with MobiLink, 585
SQL Anywhere
   as MobiLink consolidated database, 25
   MobiLink isolation levels, 129
SQL Anywhere 16 - Oracle ODBC driver
   about, 714
   MobiLink support, 714
   Unix configuration, 716
   Windows configuration, 716
SQL Anywhere consolidated database
   MobiLink, 25
SQL Server
   (see also Microsoft SQL Server)
   as MobiLink consolidated database, 13
   MobiLink data mapping, 683
SQL synchronization logic
   MobiLink, 211
   MobiLink performance, 136
SQL syntax
   MobiLink server (mlsrv16), 40
SQL-.NET data types
   MobiLink .NET synchronization logic, 501
SQL-java data types
   about, 430
SQL_TXN_READ_COMMITTED
   MobiLink isolation levels, 129
SQLType enumeration [MobiLink server .NET API]
   description, 581
SRIDs
   spatial data synchronization, 6
start classes
   DMLStartClasses option for Java, 72
   MLSStartClasses option for .NET, 71
   MobiLink .NET synchronization logic, 503
   MobiLink Java synchronization logic, 434
start_time property
   MobiLink Profiler synchronization statistics, 191
starting
   MobiLink Profiler (mlprof), 177
   MobiLink server, 27
starting the MobiLink Profiler
about, 177
statement-based scripts
uploading rows, 239
statement-based uploads
conflict detection, 116
StaticCursorLongColBuffLen
ASE, 10
statistical properties
MobiLink, 191
statistics
MobiLink, 191
STOP SYNCHRONIZATION DELETE statement
SQL Anywhere clients, 123
usage, 244
stop utility (mlstop)
syntax, 652
stopping
MobiLink server, 29
MobiLink stop utility (mlstop), 652
upload of deletes for SQL Anywhere clients, 123
stored procedures
MobiLink, 605
MobiLink stored procedure source code, 238
using to add or delete synchronization scripts, 236
using to download data, 127
stream parameters
(see also protocol options)
subsets
downloading subsets of data to remotes, 104
switches
MobiLink server (mlsrv16), 40
MobiLink user authentication utility (mluser), 654
Sybase Adaptive Server Enterprise (see Adaptive
Server Enterprise)
sync property
MobiLink Profiler synchronization statistics, 191
sync_deadlocks property
MobiLink Profiler synchronization statistics, 191
sync_errors property
MobiLink Profiler synchronization statistics, 191
sync_request property
MobiLink Profiler synchronization statistics, 191
sync_tables property
MobiLink Profiler synchronization statistics, 191
sync_warnings property
MobiLink Profiler synchronization statistics, 191
syncase.sql
about, 9
syncdb2long.sql
about, 11
synchana.sql
about, 23
synchronization
alphabetic list of scripts, 248
conflict resolution, 115
connection parameters for Profiler, 177
consolidated databases, 1
data type mappings in MobiLink, 666
deleting rows, 244
downloading rows, 242
event overview, 248
many-to-many relationships, 106
MobiLink ASE data types, 666
MobiLink character set conversion, 712
MobiLink character sets, 712
MobiLink IBM DB2 LUW data types, 675
MobiLink Microsoft SQL Server data types, 683
MobiLink MySQL data types, 690
MobiLink Oracle data types, 695
MobiLink SAP HANA data types, 704
MobiLink SAP Sybase IQ data types, 708
MobiLink system procedures, 605
MobiLink utilities, 652
performance tips, 131
process, 214
protocol options for mlsrv16, 85
restartable downloads, 124
running the MobiLink server, 27
snapshot, 102
spatial data, 6
techniques, 98
writing MobiLink scripts in .NET, 497
writing MobiLink scripts in Java, 427
writing scripts, 211
synchronization errors
handling MobiLink, 246
troubleshooting, 55
synchronization events
about, 248
about MobiLink synchronization, 248
alphabetic list of event scripts, 248
ASE begin_connection_autocommit connection
event, 276
MobiLink download, 257
MobiLink upload, 255
synchronization interval
MobiLink agent, 153
synchronization logic
MobiLink, 211
synchronization parameters
HTTP synchronization, 86
HTTPS synchronization, 86
TCP/IP synchronization, 86
synchronization properties
MobiLink Profiler, 187
synchronization scripts
.NET, 497
.NET methods, 502
about, 211
adding and deleting, 235
adding or deleting with stored procedures, 236
adding with Sybase Central, 236
connection scripts, 215
DBMS dependencies, 4
download_cursor, 243
example, 213
execution during, 214
handle_error event, 246
implementing for .NET, 498
implementing for Java, 427
Java, 427
Java methods, 431
MobiLink events, 248
parameters, 216
report_error, 248
script versions, 232
supported DBMS scripting strategies, 4
table scripts, 215
types, 215
writing scripts to download rows, 242
writing scripts to upload rows, 239
synchronization server (see MobiLink server)
synchronization stream libraries
MobiLink server deployment on 32-bit Unix, 724
MobiLink server deployment on 32-bit Windows, 718
MobiLink server deployment on 64-bit Unix, 726
MobiLink server deployment on 64-bit Windows, 721
synchronization subscriptions
(see also subscriptions)
synchronization techniques
about, 98
deleting rows, 122
failed downloads, 124
partitioning, 104
primary key pools, 112
snapshot-based synchronization, 102
stored procedures to download, 127
timestamp-based synchronization, 99
uploading rows, 239
synchronization users
MobiLink user authentication utility (mluser), 653
synchronization_statistics
connection event, 388
table event, 392
SynchronizationException class [MobiLink server .NET API]
description, 570
SynchronizationException constructor, 570
SynchronizationException class [MobiLink server Java API]
description, 480
SynchronizationException constructor, 481
SynchronizationException constructor
SynchronizationException class [MobiLink server .NET API], 570
SynchronizationException class [MobiLink server Java API], 481
synchronizing
(see also synchronization)
spatial data, 6
synchronizing data sources other than consolidated databases
about, 585
synchronizing new remotes
MobiLink file-based download, 198
synchronizing self-referencing tables
MobiLink, 129
synchronizing spatial data
named parameters, 6
Oracle considerations, 7
restrictions, 6
SRID requirements, 6
upload and download scripts, 8
syncmss.sql
about, 13
syncora.sql
about, 18
SyncRoot property
DBParameterCollection class [MobiLink server .NET API], 541
syncsa.sql
about, 25
syntax
MobiLink scripts, 248
MobiLink server (mlsrv16), 40
MobiLink stop utility (mlstop), 652
MobiLink synchronization utilities, 652
MobiLink system procedures, 605
MobiLink user authentication utility (mluser), 653
system database
MobiLink, 3
system parameters
MobiLink scripts, 216
system procedures
alphabetical list of MobiLink system procedures, 605
ml_add_cs, 609
ml_add_dcs, 610
ml_add_dts, 611
ml_add_jcs, 612
ml_add_lcs, 615
ml_add_lcs_chk, 615
ml_add_lts, 615
MobiLink, 605
system procedures for LDAP
MobiLink server, 607
system procedures to add or delete properties
MobiLink server, 607
system procedures to add or delete scripts
MobiLink server, 606
system tables
creating in MobiLink consolidated database, 2

T

table scripts
about, 215
adding .NET scripts, 611
adding Java scripts, 614
adding SQL scripts, 624
adding with Sybase Central, 237
alphabetical list of MobiLink scripts, 248
defined, 215
deleting .NET scripts, 611
deleting Java scripts, 614
deleting SQL scripts, 624
MobiLink, 212
table-level scripts
defined, 215
tables
partitioning, 104
relating consolidated tables to MobiLink remote tables, 2
tablespace capacity
IBM DB2 MobiLink consolidated databases, 12
tasks
(see also remote tasks)
TCP/IP
MobiLink mlsrv16 -x option, 86
text files
synchronizing with MobiLink, 585
Text property
LogMessage class [MobiLink server .NET API], 554
this property
DBParameterCollection class [MobiLink server .NET API], 542
threading
(see also threads)
automatic adjustment, 133
MobiLink performance, 133
threads
MobiLink worker threads and performance, 133
time changes
MobiLink, 102
time_statistics
connection event, 395
table event, 398
timeout
-tc option (mlsrv16), 75
-tf option (mlsrv16), 76
timestamp-based downloads
about, 99
timestamp-based synchronization
about, 99
download_cursor script, 100
timestamps
MobiLink download, 101
TimestampWithTimeZone class [MobiLink server Java API]
description, 482
equals method, 484
getTimeZoneOffsetHours method, 485
timeZoneoffsetMinutes method, 485

Copyright © 2014, SAP AG or an SAP affiliate company. - SAP Sybase SQL Anywhere 16.0 767
setTimeZoneoffsetHours method, 486
setTimeZoneoffsetMinutes method, 486
TimestampWithTimeZone constructor, 483
toString method, 486
toTimestampWithTimeZone method, 487
valueOf method, 487
TimestampWithTimeZone constructor
  TimestampWithTimeZone class [MobiLink server Java API], 483
TimeZoneHour property
  DateTimeWithTimeZone class [MobiLink server .NET API], 519
TimeZoneMinute property
  DateTimeWithTimeZone class [MobiLink server .NET API], 519
tips
  performance of MobiLink, 131
  synchronization techniques, 98
TLS
  (see also transport-layer security)
  MobiLink client deployment on Unix, 730
  MobiLink client deployment on Windows, 728
  MobiLink mlsrv16 -x option, 87
  MobiLink server deployment on 32-bit Unix, 724
  MobiLink server deployment on 32-bit Windows, 718
  MobiLink server deployment on 64-bit Unix, 726
  MobiLink server deployment on 64-bit Windows, 721
tools
  MobiLink Profiler marquee tool, 186
ToString method
  DateTimeWithTimeZone class [MobiLink server .NET API], 516
toString method
  TimestampWithTimeZone class [MobiLink server Java API], 486
toTimestampWithTimeZone method
  TimestampWithTimeZone class [MobiLink server Java API], 487
transactional uploads (see transaction-level uploads)
  MobiLink performance, 133
transactions
  MobiLink, 248
  MobiLink .NET synchronization logic, 501
  MobiLink Java synchronization logic, 430
translation between character sets
  MobiLink synchronization, 712
troubleshooting
  handling failed downloads, 124
  MobiLink remote data loss, 101
  MobiLink restartable downloads, 124
  MobiLink server log, 29
  MobiLink server startup, 39
  synchronization errors, 55
trusted_certificates
  MobiLink mlsrv16 -x option for HTTPS, 89
  MobiLink mlsrv16 -x option for TLS over TCP/IP, 87
tuning performance
  MobiLink , 137
Type property
  LogMessage class [MobiLink server .NET API], 554
U
  MobiLink user-defined parameter prefix, 230
ULRollbackPartialDownload function
  restartable downloads, 125
UltraLite
  deploying, 731
uni-directional synchronization
  about, 107
unique
  primary keys in MobiLink, 108
unique keys
  MobiLink, 108
unique primary keys
  generating for MobiLink using composite keys, 108
  generating for MobiLink using UUIDs, 109
  generating using key pools for MobiLink, 112
  global autoincrement for MobiLink, 109
  MobiLink, 108
Unix
  deploying dbmlsync, 730
  MobiLink server as a daemon, 31
UPDATE conflicts
  MobiLink, 115
UpdateDataReader interface [MobiLink server .NET API]
  description, 572
  SetNewRowValues method, 573
  SetOldRowValues method, 573
UpdateResultSet interface [MobiLink server Java API]
   description, 487
   setNewRowValues method, 488
   setOldRowValues method, 489
upgrading applications
   using multiple MobiLink script versions, 232
upload events
   about, 239
   MobiLink synchronization, 255
upload only synchronization
   (see also upload-only synchronization)
upload property
   MobiLink Profiler synchronization statistics, 191
upload transaction
   MobiLink, 250
upload-only and download-only synchronizations
   about, 107
upload-only synchronization
   about, 107
   required scripts, 235
upload_bytes property
   MobiLink Profiler synchronization statistics, 191
upload_deadlocks property
   MobiLink Profiler synchronization statistics, 191
upload_delete
   table event, 401
upload_deleted_rows property
   MobiLink Profiler synchronization statistics, 191
upload_errors property
   MobiLink Profiler synchronization statistics, 191
upload_fetch
   detecting conflicts, 116
   overview of conflict detection, 116
   table event, 403
upload_fetch_column_conflict
   detecting conflicts, 116
   overview of conflict detection, 116
   table event, 405
upload_insert
   table event, 407
upload_inserted_rows property
   MobiLink Profiler synchronization statistics, 191
upload_new_row_insert
   table event, 409
upload_old_row_insert
   table event, 411
upload_statistics
   connection event, 413
   table event, 417
upload_update
   detecting conflicts, 118
   overview of conflict detection, 116
   table event, 422
   using, 120
upload_updated_rows property
   MobiLink Profiler synchronization statistics, 191
upload_warnings property
   MobiLink Profiler synchronization statistics, 191
UploadData interface [MobiLink server .NET API]
   description, 573
   GetUploadedTableByName method, 574
   GetUploadedTables method, 575
UploadData interface [MobiLink server Java API]
   description, 489
   getUploadedTableByName method, 490
   getUploadedTables method, 491
UploadedTableData interface [MobiLink server .NET API]
   description, 575
   GetDeletes method, 576
   GetInserts method, 577
   GetName method, 578
   GetSchemaTable method, 579
   GetUpdates method, 579
UploadedTableData interface [MobiLink server Java API]
   description, 491
   getDeletes method, 493
   getInserts method, 494
   getMetaData method, 494
   getName method, 496
   getUpdates method, 496
uploading data from self-referencing tables
   about, 129
uploading rows
   .NET synchronization techniques, 508
   MobiLink performance, 136
   writing scripts, 239
uploads
   MobiLink scripts to upload rows, 239
   MobiLink temporarily stopping, 123
   MobiLink transaction, 250
user authentication utility (mluser)
   syntax, 653
user names
MobiLink user authentication utility (mluser), 653
user parameters
  MobiLink, 230
User property
  LogMessage class [MobiLink server .NET API], 555
user property
  MobiLink Profiler synchronization statistics, 191
user-defined parameters
  MobiLink, 230
user-defined procedures
  IBM DB2 MobiLink consolidated databases, 12
user-defined start classes
  MobiLink .NET synchronization logic, 503
  MobiLink Java synchronization logic, 434
UTC TIMESTAMP
  MobiLink, 102
utilities
  MobiLink, 652
  MobiLink stop (mlstop) syntax, 652
  MobiLink user authentication (mluser) syntax, 653
utilization graph pane
  MobiLink Profiler, 183
UUIDs
  MobiLink synchronization application, 109

V
validating
  MobiLink automatically, 200
  MobiLink custom, 202
  MobiLink file-based download, 199
validation checks
  MobiLink file-based download, 199
Value property
  DBParameter class [MobiLink server .NET API], 533
valueOf method
  TimestampWithTimeZone class [MobiLink server Java API], 487
VARBIT data type
  restrictions in ASE MobiLink consolidated databases, 10
VARCHAR data type
  MobiLink and other DBMSs, 5
Varray (Oracle)
  example, 22
  restrictions, 22
using in stored procedures, 22
VB (see Visual Basic)
 verbosity
  MobiLink performance, 135
  MobiLink server option (mlsrv16) -v, 79
version option
  MobiLink mlsrv16 -x option for OE, 91
version property
  MobiLink Profiler synchronization statistics, 191
version protocol option
  MobiLink mlsrv16 -x option for HTTP, 88
  MobiLink mlsrv16 -x option for HTTPS, 89
versions
  about MobiLink synchronization scripts, 232
  adding script versions, 234
  removing script versions, 234
viewing MobiLink logs
  about, 30
Visual Basic
  MobiLink synchronization scripts, 497
  support in MobiLink .NET, 497
Visual Studio
  MobiLink synchronization scripts, 497

W
WARNING variable
  LogMessage class [MobiLink server Java API], 462
WarningListener event
  ServerContext interface [MobiLink server .NET API], 566
web servers
  synchronizing with MobiLink, 585
web services
  synchronizing with MobiLink, 585
WebLogic
  MobiLink and, 585
Windows CE (see Windows Mobile)
Windows Mobile Device Center (see Microsoft ActiveSync)
worker threads
  automatic adjustment, 133
  MobiLink, 137
  MobiLink performance, 133
writing
  .NET synchronization logic, 497
  Java synchronization logic, 427
X
Xusage.txt
  location, 72

Y
Year property
  DateTimeWithTimeZone class [MobiLink server .NET API], 520