



**PUBLIC**

SQL Anywhere - MobiLink

Document Version: 17.01.0 – 2021-10-15

# MobiLink Server Administration

# Content

- 1    **MobiLink - Server Administration.** . . . . . **7****
- 1.1    **MobiLink Deployment.** . . . . . **8**
  - MobiLink Server Deployment. . . . . 9
  - Microsoft Windows 32-Bit Applications. . . . . 10
  - Microsoft Windows 64-Bit Applications. . . . . 13
  - 64-Bit Applications on UNIX and Linux. . . . . 16
  - SQL Anywhere MobiLink Client Deployment. . . . . 18
  - Microsoft Windows Applications. . . . . 18
  - Applications on UNIX, Linux, and macOS. . . . . 20
  - UltraLite MobiLink Client Deployment. . . . . 21
- 1.2    **MobiLink Server.** . . . . . **21**
  - Required Privileges for MobiLink Server. . . . . 23
  - MobiLink Connectivity. . . . . 24
  - MobiLink Server Shutdown. . . . . 24
  - MobiLink Server Logging. . . . . 25
  - MobiLink Server Use Outside the Current Session. . . . . 29
  - MobiLink Server in a Server Farm. . . . . 37
  - Troubleshooting MobiLink Server Startup. . . . . 38
- 1.3    **MobiLink Server Options.** . . . . . **40**
  - mlsrv17 Syntax. . . . . 44
  - @data mlsrv17 Option. . . . . 50
  - a mlsrv17 Option. . . . . 51
  - b mlsrv17 Option. . . . . 51
  - bn mlsrv17 Option. . . . . 52
  - c mlsrv17 Option. . . . . 53
  - ca mlsrv17 Option. . . . . 54
  - cinit mlsrv17 Option. . . . . 54
  - cn mlsrv17 Option. . . . . 55
  - cr mlsrv17 Option. . . . . 56
  - cs mlsrv17 Option. . . . . 56
  - ct mlsrv17 Option. . . . . 57
  - dl mlsrv17 Option. . . . . 57
  - dr mlsrv17 Option. . . . . 58
  - ds mlsrv17 Option. . . . . 58
  - dsd mlsrv17 Option. . . . . 59
  - dt mlsrv17 Option. . . . . 60

-e mlsrv17 Option. . . . .	61
-esu mlsrv17 Option. . . . .	62
-et mlsrv17 Option. . . . .	63
-fips mlsrv17 Option. . . . .	63
-ftr mlsrv17 Option. . . . .	64
-ftru mlsrv17 Option. . . . .	65
-lsc mlsrv17 Option. . . . .	65
-nc mlsrv17 Option. . . . .	66
-ncs mlsrv17 Option. . . . .	67
-ncsd mlsrv17 Option. . . . .	68
-ncsp mlsrv17 Option. . . . .	69
-notifier mlsrv17 Option. . . . .	69
-o mlsrv17 Option. . . . .	70
-on mlsrv17 Option. . . . .	71
-oq mlsrv17 Option. . . . .	72
-os mlsrv17 Option. . . . .	73
-ot mlsrv17 Option. . . . .	74
-ppv mlsrv17 Option. . . . .	74
-q mlsrv17 Option. . . . .	79
-r mlsrv17 Option. . . . .	79
-rd mlsrv17 Option. . . . .	79
-rp mlsrv17 Option. . . . .	80
-rrp mlsrv17 Option. . . . .	81
-s mlsrv17 Option. . . . .	81
-sl dnet mlsrv17 Option. . . . .	82
-sl java mlsrv17 Option. . . . .	84
-sm mlsrv17 Option. . . . .	85
-tc mlsrv17 Option. . . . .	86
-tf mlsrv17 Option. . . . .	87
-ts mlsrv17 Option. . . . .	88
-tx mlsrv17 Option. . . . .	89
-ud mlsrv17 Option. . . . .	90
-ui mlsrv17 Option. . . . .	91
-ux mlsrv17 Option. . . . .	91
-v mlsrv17 Option. . . . .	92
-w mlsrv17 Option. . . . .	96
-wm mlsrv17 Option. . . . .	97
-wn mlsrv17 Option. . . . .	98
-wu mlsrv17 Option. . . . .	98
-x mlsrv17 Option. . . . .	99
-zf mlsrv17 Option. . . . .	107

	-zp mlsrv17 Option. . . . .	108
	-zs mlsrv17 Option. . . . .	108
	-zt mlsrv17 Option. . . . .	109
	-zu mlsrv17 Option. . . . .	109
	-zup mlsrv17 Option. . . . .	110
	-zus mlsrv17 Option. . . . .	111
	-zw mlsrv17 Option. . . . .	111
	-zwd mlsrv17 Option. . . . .	112
	-zwe mlsrv17 Option. . . . .	113
1.4	Synchronization Techniques. . . . .	113
	Implementing Timestamp-based Downloads. . . . .	115
	Snapshot Synchronization. . . . .	119
	Partitioned Rows Among Remote Databases. . . . .	121
	Upload-only and Download-only Synchronizations. . . . .	125
	Unique Primary Keys. . . . .	126
	Conflict Handling Overview. . . . .	133
	Deletes. . . . .	142
	Failed Downloads. . . . .	144
	Download Acknowledgement. . . . .	147
	Result Sets from Stored Procedure Calls. . . . .	147
	Self-referencing Tables. . . . .	149
	MobiLink Isolation Levels. . . . .	149
1.5	MobiLink Consolidated Databases. . . . .	152
	How Remote Tables Relate to Consolidated Tables. . . . .	154
	Consolidated Database Setup. . . . .	155
	RDBMS-Dependent Synchronization Scripts. . . . .	157
	Synchronization of Spatial Data. . . . .	158
	Adaptive Server Enterprise Consolidated Database. . . . .	163
	IBM DB2 LUW Consolidated Database. . . . .	165
	Microsoft SQL Server and Microsoft Azure Consolidated Databases. . . . .	168
	MySQL Consolidated Database. . . . .	170
	Oracle Consolidated Database. . . . .	173
	SAP HANA Consolidated Database. . . . .	179
	SQL Anywhere Consolidated Database. . . . .	181
	SAP IQ Consolidated Database. . . . .	182
1.6	MobiLink Performance. . . . .	183
	Test to Improve Performance. . . . .	186
	Avoid Contention. . . . .	186
	Use Multithreaded Network Processing. . . . .	187
	Use an Optimal Number of Database Worker Threads. . . . .	187
	Automatic Adjustment of Database Worker Threads. . . . .	187

	Use Smaller Upload Transactions. . . . .	188
	Avoid Synchronizing Unnecessary BLOBs. . . . .	188
	Set the Maximum Number of Database Connections. . . . .	188
	Have Enough Physical Memory. . . . .	188
	Use Enough Processing Power. . . . .	189
	Optimize Script Execution. . . . .	189
	Use Minimum Logging Verbosity. . . . .	189
	Plan for Operating System Limitations. . . . .	189
	Java or .NET vs. SQL Synchronization Logic. . . . .	190
	Priority Synchronization. . . . .	190
	Download Only the Rows You Need. . . . .	191
	Only Synchronize When You Need to. . . . .	191
	For Large Uploads, Estimate the Number of Rows. . . . .	191
	Use Background Synchronization. . . . .	191
	Key Factors Influencing MobiLink Performance. . . . .	191
	MobiLink Performance Monitoring. . . . .	195
1.7	MobiLink Client/Server Communications Encryption. . . . .	196
	End-to-end Encryption. . . . .	196
	Starting the MobiLink Server with Transport Layer Security. . . . .	197
	MobiLink Client Configuration to Use Transport Layer Security. . . . .	198
1.8	Manage Remote Databases. . . . .	202
	Central Administration Concepts. . . . .	204
	MobiLink Agents. . . . .	207
	Remote Tasks. . . . .	219
	Deployment and Configuration. . . . .	245
1.9	MobiLink Profiler. . . . .	247
	Starting the MobiLink Profiler (Administration Tools). . . . .	249
	MobiLink Profiler (mlprof) on the Command Line. . . . .	250
	Starting a Profiling Session. . . . .	250
	Ending a Profiling Session. . . . .	252
	Opening or Deleting a Previous Profiling Session. . . . .	253
	The Profiling Database. . . . .	253
	MobiLink Profiler Interface. . . . .	254
	Statistic Customization. . . . .	263
	Using the Profiling Database. . . . .	264
	MobiLink Synchronization Statistical Properties. . . . .	267
1.10	MobiLink File-based Download. . . . .	270
	File-Based Download Setup. . . . .	271
	Validation Checks. . . . .	275
	File-Based Download Examples. . . . .	278
1.11	The Relay Server Reverse Proxy. . . . .	287

1.12	MobiLink Events. . . . .	288
	Synchronization Scripts. . . . .	288
	Synchronization Events. . . . .	332
1.13	MobiLink Server APIs. . . . .	525
	Synchronization Script Writing in Java. . . . .	526
	MobiLink Server Java API Reference. . . . .	540
	Synchronization Scripts in Microsoft .NET. . . . .	541
	MobiLink Server .NET API Reference. . . . .	557
	Direct Row Handling. . . . .	558
1.14	MobiLink Reference. . . . .	569
	MobiLink Replay C++ Callbacks. . . . .	570
	MobiLink Server System Procedures. . . . .	582
	MobiLink Utilities. . . . .	647
	MobiLink Data Mappings Between Remote and Consolidated Databases. . . . .	662
	Character Set Considerations. . . . .	708
	ODBC Drivers for MobiLink. . . . .	710

# 1 MobiLink - Server Administration

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.

## i Note

Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were previously performed in the Monitor can be performed in the SQL Anywhere Cockpit. See [SQL Anywhere Monitor Non-GUI User Guide](#).

## In this section:

### [MobiLink Deployment \[page 8\]](#)

Deploying MobiLink applications involves the following activities:

### [MobiLink Server \[page 21\]](#)

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.

### [MobiLink Server Options \[page 40\]](#)

The following options are available for use with the MobiLink server.

### [Synchronization Techniques \[page 113\]](#)

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.

### [MobiLink Consolidated Databases \[page 152\]](#)

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 Performance \[page 183\]](#)

The following is a list of suggestions to help you get the best performance out of MobiLink.

### [MobiLink Client/Server Communications Encryption \[page 196\]](#)

You can encrypt MobiLink client/server communication using transport layer security.

### [Manage Remote Databases \[page 202\]](#)

You can centrally manage remote databases involved in MobiLink synchronization using the MobiLink plug-in SQL Central.

### [MobiLink Profiler \[page 247\]](#)

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.

### [MobiLink File-based Download \[page 270\]](#)

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.

#### [The Relay Server Reverse Proxy \[page 287\]](#)

The Relay Server is a reverse proxy that enables secure, load-balanced communication between mobile devices and backend servers through a web server. Supported backend servers include MobiLink, SAP Mobile Server, SAP Afaria, and SAP Mobile Office.

#### [MobiLink Events \[page 288\]](#)

The synchronization process has multiple steps and a unique event identifies each step. You control the synchronization process by writing scripts associated with some of these events.

#### [MobiLink Server APIs \[page 525\]](#)

MobiLink synchronization scripts can be written in SQL, in Java (using the MobiLink server API for Java) or in .NET (using the MobiLink server API for .NET).

#### [MobiLink Reference \[page 569\]](#)

Many useful tools and resources are available to help you use MobiLink.

## 1.1 MobiLink Deployment

Deploying MobiLink applications involves the following activities:

- Deploy the MobiLink server into a production setting.
- Deploy any SQL Anywhere MobiLink clients.
- Deploy any UltraLite MobiLink clients.

The *Deploy Synchronization Model Wizard* can help with your deployment on Microsoft Windows.

### 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.

#### In this section:

##### [MobiLink Server Deployment \[page 9\]](#)

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.

##### [Microsoft Windows 32-Bit Applications \[page 10\]](#)

All directories are relative to %SQLANY17%.

##### [Microsoft Windows 64-Bit Applications \[page 13\]](#)

All directories are relative to %SQLANY17%.

##### [64-Bit Applications on UNIX and Linux \[page 16\]](#)

All directories are relative to \$SQLANY17.

##### [SQL Anywhere MobiLink Client Deployment \[page 18\]](#)



Keep the following notes in mind for SQL Anywhere MobiLink client deployment.

#### [Microsoft Windows Applications \[page 18\]](#)

All directories are relative to %SQLANY17%. There are 64-bit versions of these files in the Bin64 directory.

#### [Applications on UNIX, Linux, and macOS \[page 20\]](#)

All directories are relative to \$SQLANY17.

#### [UltraLite MobiLink Client Deployment \[page 21\]](#)

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.

## Related Information

[The Deployment Wizard for Windows](#)

### 1.1.1 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 and Linux, environment variables must be set for the system to locate SQL Anywhere applications and libraries. Use the appropriate file for your shell, either `sa_config.sh` or `sa_config.csh` 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`, `SQLANY17`, and `SQLANYSAMP17`.
- On Microsoft 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 `%SQLANY17%\Bin32` for 32-bit environments or `%SQLANY17%\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 (SQL Central and the MobiLink Profiler), you must have JRE 1.8.0 installed.

## Related Information

[The Deployment Wizard for Windows Administration Tool Deployment](#)

### 1.1.2 Microsoft Windows 32-Bit Applications

All directories are relative to %SQLANY17%.

Description	Microsoft Windows files
MobiLink server	The MobiLink server is only available as a 64-bit application.
Language library	<ul style="list-style-type: none"><li>Bin32\dblgen17.dll<sup>1</sup></li></ul>
Java synchronization logic	<ul style="list-style-type: none"><li>Java\activation.jar<sup>2</sup></li><li>Java\imap.jar<sup>2</sup></li><li>Java\jodbc4.jar</li><li>Java\mailapi.jar<sup>2</sup></li><li>Java\mlscript.jar</li><li>Java\mlsupport.jar</li><li>Java\pop3.jar<sup>2</sup></li><li>Java\smtp.jar<sup>2</sup></li><li>Bin32\dbjodbc17.dll</li><li>Bin32\mljodbc17.dll</li></ul>
.NET synchronization logic	<ul style="list-style-type: none"><li>MobiLink\Setup\Dnet\mlDomConfig.xml</li><li>Assembly\V2\Sap.MobiLink.dll</li><li>Assembly\V2\Sap.MobiLink.Script.dll</li><li>Assembly\V2\Sap.MobiLink.Script.xml</li><li>Bin32\mlDomConfig.xsd</li></ul>
Encrypted communications <sup>3</sup>	<ul style="list-style-type: none"><li>Bin32\dbrsa17.dll</li><li>Bin32\dbfips17.dll</li><li>Bin32\sapcrypto.dll</li><li>Bin32\sapcryptofips.dll</li><li>Bin32\slcryptokernel.dll</li><li>Bin32\slcryptokernel.dll.sha256</li><li>Bin32\msvcr90.dll</li></ul>
Setup scripts (deploy the ones for your consolidated database)	<ul style="list-style-type: none"><li>MobiLink\Setup\ </li><li>MobiLink\Upgrade\ </li></ul>

Description	Microsoft Windows files
mluser utility	<ul style="list-style-type: none"> <li>• Bin32\mluser.exe</li> <li>• Bin32\dbicu17.dll</li> <li>• Bin32\dbicudt17.dll</li> </ul>
mlstop utility	<ul style="list-style-type: none"> <li>• Bin32\mlstop.exe</li> <li>• Bin32\dbicu17.dll</li> </ul>
mlreplay utility <sup>6</sup>	<ul style="list-style-type: none"> <li>• Bin32\mlreplay.exe</li> <li>• Bin32\mlgenreplayapi.exe</li> </ul>
MobiLink arbiter	<ul style="list-style-type: none"> <li>• Bin32\dserv17.dll</li> <li>• Bin32\mlarb17.exe</li> <li>• Bin32\mlarb17.lic</li> <li>• Bin32\mlarbiter.bat</li> <li>• MobiLink\mlarbiter.control</li> <li>• mlarbstop.exe</li> </ul>
MobiLink Profiler	<ul style="list-style-type: none"> <li>• Java\mlprof.jar</li> <li>• Java\mlstream.jar</li> <li>• Java\JComponents.jar</li> <li>• Java\jsyblib1700.jar</li> <li>• Bin32\jsyblib1700.dll</li> <li>• Bin32\mlprof.exe</li> <li>• Bin32\mljstrm17.dll</li> </ul> <p data-bbox="804 1245 1190 1267">For security with the MobiLink Profiler:<sup>3</sup></p> <ul style="list-style-type: none"> <li>• Bin32\mlcrsa17.dll</li> <li>• Bin32\mlcrsafips17.dll</li> <li>• Bin32\mlczlib17.dll</li> </ul>
Online help for the MobiLink 17 plug-in and MobiLink Profiler	<ul style="list-style-type: none"> <li>• Documentation\en\htmlhelp\sqlanywhere_en17.chm<sup>1</sup></li> <li>• Documentation\en\htmlhelp\sqlanywhere_en17.map<sup>1</sup></li> </ul>

Description	Microsoft Windows files
Notifier	<ul style="list-style-type: none"> <li>• Java\jodbc4.jar</li> <li>• Java\mlnotif.jar</li> <li>• Java\mlscript.jar</li> <li>• Bin32\mljodbc17.dll</li> <li>• Bin32\mljstrm17.dll</li> <li>• Bin32\mlsmtp17.dll</li> </ul> <p>For security with the Notifier:<sup>3</sup></p> <ul style="list-style-type: none"> <li>• Bin32\mlcrsa17.dll</li> <li>• Bin32\mlcrsafips17.dll</li> <li>• Bin32\mlczlib17.dll</li> </ul>
Relay Server Outbound Enabler	<ul style="list-style-type: none"> <li>• Bin32\rsoe2.exe</li> </ul> <p>For security with the Outbound Enabler:</p> <ul style="list-style-type: none"> <li>• Bin32\mlcrsa17.dll</li> </ul>

<sup>1</sup> For German, Japanese, and Chinese editions, use `dblgde17.dll`, `dblgja17.dll` and `dblgzh17.dll` respectively.

<sup>2</sup> If you are redistributing an application, you must obtain these files directly from Oracle.

<sup>3</sup> The non-certified version of RSA encryption is included with SQL Anywhere. FIPS-certified encryption is separately licensed.

<sup>4</sup> To compile generated code, any files prefixed with `mlreplay` in `SDK/Include` are required.

## Related Information

[Separately Licensed Components](#)

## 1.1.3 Microsoft Windows 64-Bit Applications

All directories are relative to `%SQLANY17%`.

Description	Microsoft Windows files
MobiLink server	<ul style="list-style-type: none"> <li>Bin64\mlodbc17.dll</li> <li>Bin64\mlsrv17.exe</li> <li>Bin64\mlserv17.dll</li> <li>Bin64\mlsrv17.lic</li> <li>Bin64\mlsql17.dll</li> <li>Bin64\dbicu17.dll</li> <li>Bin64\dbicudt17.dll</li> </ul>
Language library	<ul style="list-style-type: none"> <li>Bin64\dblgen17.dll<sup>1</sup></li> </ul>
Java synchronization logic	<ul style="list-style-type: none"> <li>Java\activation.jar<sup>2</sup></li> <li>Java\imap.jar<sup>2</sup></li> <li>Java\jodbc4.jar</li> <li>Java\mailapi.jar<sup>2</sup></li> <li>Java\mlscript.jar</li> <li>Java\mlsupport.jar</li> <li>Java\pop3.jar<sup>2</sup></li> <li>Java\smtp.jar<sup>2</sup></li> <li>Bin64\mljava17.dll</li> <li>Bin64\dbjodbc17.dll</li> <li>Bin64\mljodbc17.dll</li> </ul>
.NET synchronization logic	<ul style="list-style-type: none"> <li>MobiLink\Setup\Dnet\mlDomConfig.xml</li> <li>Bin64\mldnet17.dll</li> <li>Bin64\dnetodbc17.dll</li> <li>Assembly\V3.5\Sap.MobiLink.dll</li> <li>Assembly\V3.5\Sap.MobiLink.Script.dll</li> <li>Assembly\V3.5\Sap.MobiLink.Script.xml</li> <li>Bin64\mlDomConfig.xsd</li> </ul>
Encrypted communications <sup>3</sup>	<ul style="list-style-type: none"> <li>Bin64\dbrsa17.dll</li> <li>Bin64\dbfips17.dll</li> <li>Bin32\sapcrypto.dll</li> <li>Bin32\sapcryptofips.dll</li> <li>Bin32\slcryptokernel.dll</li> <li>Bin32\slcryptokernel.dll.sha256</li> <li>Bin64\msvcr100.dll</li> </ul>
Setup scripts (deploy the ones for your consolidated database)	<ul style="list-style-type: none"> <li>MobiLink\Setup\</li> <li>MobiLink\Upgrade\</li> </ul>

Description	Microsoft Windows files
mluser utility	<ul style="list-style-type: none"> <li>• Bin64\mluser.exe</li> <li>• Bin64\mlodbc17.dll</li> <li>• Bin64\dbicu17.dll</li> <li>• Bin64\dbicudt17.dll</li> </ul>
mlstop utility	<ul style="list-style-type: none"> <li>• Bin64\mlstop.exe</li> <li>• Bin64\dbicu17.dll</li> </ul>
mlreplay utility <sup>6</sup>	<ul style="list-style-type: none"> <li>• Bin64\mlreplay.exe</li> <li>• Bin64\mlgenreplayapi.exe</li> </ul>
MobiLink arbiter	<ul style="list-style-type: none"> <li>• Bin64\dbserv17.dll</li> <li>• Bin64\mlarb17.exe</li> <li>• Bin64\mlarb17.lic</li> <li>• Bin64\mlarbiter.bat</li> <li>• MobiLink\mlarbiter.control</li> <li>• mlarbstop.exe</li> </ul>
MobiLink Profiler	<ul style="list-style-type: none"> <li>• Java\mlprof.jar</li> <li>• Java\mlstream.jar</li> <li>• Java\JComponents1700.jar</li> <li>• Java\jsyblib1700.jar</li> <li>• Bin64\jsyblib1700.dll</li> <li>• Bin64\mlprof.exe</li> <li>• Bin64\mljstrm17.dll</li> </ul> <p data-bbox="804 1285 1190 1308">For security with the MobiLink Profiler:<sup>3</sup></p> <ul style="list-style-type: none"> <li>• Bin64\mlcrsa17.dll</li> <li>• Bin64\mlcrsafips17.dll</li> <li>• Bin64\mlczlib17.dll</li> </ul>
Online help for the MobiLink 17 plug-in and MobiLink Profiler	<ul style="list-style-type: none"> <li>• Documentation\en\htmlhelp\sqlanywhere_en17.chm<sup>1</sup></li> <li>• Documentation\en\htmlhelp\sqlanywhere_en17.map<sup>1</sup></li> </ul>

Description	Microsoft Windows files
Notifier	<ul style="list-style-type: none"> <li>• Java\jodbc4.jar</li> <li>• Java\mlnotif.jar</li> <li>• Java\mlscript.jar</li> <li>• Java\jsyblib1700.jar</li> <li>• Bin64\mljodbc17.dll</li> <li>• Bin64\mljstrm17.dll</li> <li>• Bin64\mlsmtp17.dll</li> </ul> <p>For security with the Notifier:<sup>3</sup></p> <ul style="list-style-type: none"> <li>• Bin64\mlcrsa17.dll</li> <li>• Bin64\mlcrsafips17.dll</li> <li>• Bin64\mlczlib17.dll</li> </ul>
Relay Server Outbound Enabler	<ul style="list-style-type: none"> <li>• Bin64\rsoe2.exe</li> </ul> <p>For security with the Outbound Enabler:</p> <ul style="list-style-type: none"> <li>• Bin64\mlcrsa17.dll</li> </ul>

<sup>1</sup> For German, Japanese, and Chinese editions, use `dblgde17.dll`, `dblgja17.dll` and `dblgzh17.dll` respectively.

<sup>2</sup> If you are redistributing an application, you must obtain these files directly from Oracle.

<sup>3</sup> The non-certified version of RSA encryption is included with SQL Anywhere. FIPS-certified encryption is separately licensed.

<sup>4</sup> To compile generated code, any files prefixed with `mlreplay` in `SDK/Include` are required.

## Related Information

[Separately Licensed Components](#)

[Microsoft Windows 32-Bit Applications \[page 10\]](#)

## 1.1.4 64-Bit Applications on UNIX and Linux

All directories are relative to `$$SQLANY17`.

Description	UNIX files
MobiLink server	<ul style="list-style-type: none"><li>• <code>bin64/mlsrv17</code></li><li>• <code>bin64/libmlserv17_r.so</code></li><li>• <code>bin64/mlsrv17.lic</code></li><li>• <code>lib64/libdbodm17.so<sup>3</sup></code></li><li>• <code>lib64/libmlodbc17_r.so<sup>3</sup></code></li><li>• <code>lib64/libmlsql17_r.so<sup>3</sup></code></li><li>• <code>lib64/libdbtasks17_r.so<sup>3</sup></code></li><li>• <code>lib64/libdbicu17_r.so<sup>3</sup></code></li><li>• <code>lib64/libdbicudt17_r.so<sup>3</sup></code></li><li>• <code>lib64/libdbodbcinst17_r.so<sup>3</sup></code></li></ul>
Language library	<ul style="list-style-type: none"><li>• <code>res/dblgen17.res<sup>1</sup></code></li></ul>
Java synchronization logic	<ul style="list-style-type: none"><li>• <code>java/activation.jar<sup>2</sup></code></li><li>• <code>java/imap.jar<sup>2</sup></code></li><li>• <code>java/jodbc4.jar</code></li><li>• <code>java/mailapi.jar<sup>2</sup></code></li><li>• <code>java/mlscript.jar</code></li><li>• <code>java/mlsupport.jar</code></li><li>• <code>java/pop3.jar<sup>2</sup></code></li><li>• <code>java/smtp.jar<sup>2</sup></code></li><li>• <code>lib64/libmljava17_r.so<sup>3</sup></code></li><li>• <code>lib64/libmljodbc17.so<sup>3</sup></code></li></ul>
.NET synchronization logic	<ul style="list-style-type: none"><li>• Not applicable</li></ul>
Encrypted communications <sup>4</sup>	<ul style="list-style-type: none"><li>• <code>lib64/libdbrsa17_r.so<sup>3</sup></code></li><li>• <code>lib64/libdbfips17_r.so</code> (Linux only)</li><li>• <code>lib64/libsapcryptofips.so</code> (Linux only)</li><li>• <code>lib64/libslcryptokernel.so</code> (Linux only)</li><li>• <code>lib64/libslcryptokernel.so.sha25</code> (Linux only)</li></ul>
Setup scripts (deploy the ones for your consolidated database)	<ul style="list-style-type: none"><li>• <code>mobilink/setup</code></li><li>• <code>mobilink/upgrade</code></li></ul>
mluser utility	<ul style="list-style-type: none"><li>• <code>bin64/mluser</code></li><li>• <code>lib64/libmlodbc17_r.so<sup>3</sup></code></li><li>• <code>lib64/libdbicu17.so<sup>3</sup></code></li><li>• <code>lib64/libdbicudt17.so<sup>3</sup></code></li></ul>



Description	UNIX files
mlstop utility	<ul style="list-style-type: none"> <li>bin64/mlstop</li> <li>lib64/libdbicu17.so<sup>3</sup></li> </ul>
mlreplay utility <sup>6</sup>	<ul style="list-style-type: none"> <li>bin64/mlreplay</li> <li>bin64/mlgenreplayapi</li> </ul>
MobiLink arbiter	<ul style="list-style-type: none"> <li>bin64/libdbserv17_r.so</li> <li>bin64/mlarb17</li> <li>bin64/mlarb17.lic</li> <li>bin64/mlarbiter.sh</li> <li>mobilink/mlarbiter.control</li> </ul>
MobiLink Profiler	<ul style="list-style-type: none"> <li>bin64/mlprof</li> <li>java/mlprof.jar</li> <li>java/mlstream.jar</li> <li>java/JComponents1700.jar</li> <li>java/jsyblib1700.jar</li> <li>lib64/libjsyblib1700_r.so<sup>3</sup></li> </ul> <p>For security with the Profiler:</p> <ul style="list-style-type: none"> <li>lib64/libmlcrsa17_r.so</li> <li>lib64/libmlcrsafips17_r.so</li> <li>lib64/libmlczlib17_r.so</li> </ul>
Notifier	<ul style="list-style-type: none"> <li>java/activation.jar<sup>2</sup></li> <li>java/jodbc4.jar</li> <li>java/mlnotif.jar</li> <li>java/mlscript.jar</li> <li>lib64/libmlsmtp17.so</li> </ul>
Relay Server Outbound Enabler	<ul style="list-style-type: none"> <li>bin64/rsoe2</li> </ul> <p>For security with the Outbound Enabler:</p> <ul style="list-style-type: none"> <li>lib64/libmlcrsa17_r.so<sup>3</sup></li> </ul>

<sup>1</sup> For German, Japanese, and Chinese editions, use `dblgede17.res`, `dblgja17.res` and `dblqzh17.res` respectively.

<sup>2</sup> If you are redistributing an application, you must obtain these files directly from Oracle.

<sup>3</sup> For Solaris SPARC and Linux, the file extension is `.so`. For IBM AIX (deprecated), the file extension is `.a`.

<sup>4</sup> The non-certified version of RSA encryption is included with SQL Anywhere. FIPS-certified encryption is separately licensed.

<sup>5</sup> To compile generated code, any files prefixed with `mlreplay` in `sdk/include` are required.

## Related Information

[Separately Licensed Components](#)

### 1.1.5 SQL Anywhere MobiLink Client Deployment

Keep the following notes in mind for SQL Anywhere MobiLink client deployment.

- For SQL Anywhere clients, you need to deploy a SQL Anywhere database server and the MobiLink client.
- 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 files, place them in the same directory structure unless otherwise noted.

## Related Information

[Database and Application Deployment](#)

[Administration Tool Deployment](#)

[The Deployment Wizard for Windows](#)

### 1.1.6 Microsoft Windows Applications

All directories are relative to `%SQLANY17%`. There are 64-bit versions of these files in the `Bin64` directory.

Description	Microsoft Windows files
MobiLink synchronization client (dbmlsync)	<ul style="list-style-type: none"><li>• <code>Bin32\dbcon17.dll</code></li><li>• <code>Bin32\dbicu17.dll</code><sup>2</sup></li><li>• <code>Bin32\dblgen17.dll</code><sup>1</sup></li><li>• <code>Bin32\dblib17.dll</code></li><li>• <code>Bin32\dbmlsync.exe</code></li><li>• <code>Bin32\dbtool17.dll</code></li><li>• <code>Bin32\sapcrypto.dll</code></li></ul>
Dbmlsync C++ API or the SQL SYNCHRONIZE statement	<ul style="list-style-type: none"><li>• MobiLink synchronization client files</li><li>• <code>Bin32\dbmlsynccli17.dll</code></li></ul>
Dbmlsync .NET API	<ul style="list-style-type: none"><li>• MobiLink synchronization client files</li><li>• <code>Assembly\V3.5\Sap.MobiLink.Client.dll</code></li></ul>

Description	Microsoft Windows files
Encrypted communications <sup>4</sup>	<p>Files for non-FIPS encryption:</p> <ul style="list-style-type: none"> <li>• Bin32\dbrsa17.dll</li> <li>• Bin32\mlcrsa17.dll</li> </ul> <p>Files FIPS encryption:</p> <ul style="list-style-type: none"> <li>• Bin32\dbrsa17.dll</li> <li>• Bin32\mlcrsafips17.dll</li> <li>• Bin32\sapcrypto.dll</li> <li>• Bin32\sapcryptofips.dll</li> <li>• Bin32\slcryptokernel.dll</li> <li>• Bin32\slcryptokernel.dll.sha256</li> <li>• Bin32\msvcr90.dll<sup>5</sup></li> </ul>
Listener	<ul style="list-style-type: none"> <li>• Bin32\dblgen17.dll<sup>1</sup></li> <li>• Bin32\dblsln.exe</li> <li>• Bin32\lsn_udp17.dll</li> </ul>
MobiLink Agent (required for central administration of remote databases)	<p>To manage a SQL Anywhere remote database, the following files must be on the remote device:</p> <ul style="list-style-type: none"> <li>• sacrypto.dll</li> <li>• dbrsa17.dll</li> <li>• dbcon17.dll</li> <li>• <a href="#">dbeng17.exe</a><sup>3</sup></li> <li>• <a href="#">dbeng17.lic</a><sup>3</sup></li> <li>• dbghelp.dll</li> <li>• dbicu17.dll</li> <li>• dbicudt17.dll<sup>3</sup></li> <li>• dblgen17.dll<sup>1</sup></li> <li>• dblib17.dll</li> <li>• dbmlsync.exe</li> <li>• dbmlsynccli17.dll</li> <li>• dbscript17.dll</li> <li>• <a href="#">dbsrv17.exe</a></li> <li>• <a href="#">dbsrv17.lic</a></li> <li>• dbtool17.dll</li> <li>• mlagent.exe</li> <li>• mlstop.exe</li> <li>• Bin32\sapcrypto.dll</li> <li>• mlasadapt17.dll</li> </ul> <p>Certain features of SQL Anywhere may need extra files to deploy.</p>

<sup>1</sup> For German, Japanese, and Chinese editions, use `dblgede17.dll`, `dblaja17.dll` and `dblgh17.dll` respectively.

<sup>2</sup> Not required if the database is initialized with `dbinit -zn UTF8BIN`.

<sup>3</sup> For Microsoft Windows operating systems.

<sup>4</sup> The non-certified version of RSA encryption is included with SQL Anywhere. FIPS-certified encryption is separately licensed.

<sup>5</sup> The 64-bit version of this file is called `msvcr100.dll`.

## Related Information

[Database and Application Deployment Initialization Utility \(dbinit\)](#)

## 1.1.7 Applications on UNIX, Linux, and macOS

All directories are relative to `SQLANY17`.

Description	UNIX files
MobiLink synchronization client (dbmlsync)	<ul style="list-style-type: none"><li><code>bin32/dbmlsync</code></li><li><code>res/dblgen17.res</code></li><li><code>lib32/libdbicu17_r.so</code><sup>1</sup></li><li><code>lib32/libdblib17_r.so</code><sup>1</sup></li><li><code>lib32/libdbtool17_r.so</code><sup>1</sup></li></ul>
Dbmlsync C++ API or the SQL SYNCHRONIZE statement	<ul style="list-style-type: none"><li>MobiLink synchronization client files</li><li><code>lib32/libdbmlsynccli17_r.so</code></li></ul>
Encrypted communications <sup>2</sup>	<ul style="list-style-type: none"><li><code>lib32/libmlcrsa17_r.so</code><sup>1</sup></li><li><code>lib32/libmlcrsafips17_r.so</code> (Linux only)</li><li><code>lib32/libdbcrypto.so</code> (Linux only)</li><li><code>lib32/libdbssl.so</code> (Linux only)</li></ul>

Support for the 64-bit MobiLink synchronization client is located in the `bin64/lib64` directories. For macOS, the file extension is `.dylib`. Only 64-bit software is supported for macOS.

<sup>1</sup>For Linux, the file extension is `.so`.

<sup>2</sup> The non-certified version of RSA encryption is included with SQL Anywhere. FIPS-certified encryption is separately licensed.

## Related Information

[Separately Licensed Components](#)

### 1.1.8 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.

## Related Information

[UltraLite Deployment](#)

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

[Lesson 5: Building and Deploying the Application](#)

## 1.2 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.

### **i** Note

The `mlsrv17` 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.

To start the MobiLink server, run `mlsrv17`. Use the `-c` option to specify the ODBC connection parameters for your consolidated database.

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.

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 Microsoft 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).

## Example

The following command starts the MobiLink server using the ODBC data source *SQL Anywhere 17 CustDB* to identify the consolidated database. Enter the entire command on one line.

```
mksrv17 -c "DSN=SQL Anywhere 17 CustDB;UID=ml_server;PWD=sql" -zs MyServer -o
mksrv.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 message log file should be named `mksrv.log`. The contents of `mksrv.log` are verbose because of the `-vcr` option. The `-x` option opens a port for version 10 and later clients.

You can also start the MobiLink server as a Microsoft Windows service or UNIX or Linux daemon.

### In this section:

#### [Required Privileges for MobiLink Server \[page 23\]](#)

You must specify a database user for the MobiLink server to connect to the database server. You specify the database user with the `mksrv17 -c` option or in the ODBC data source.

#### [MobiLink Connectivity \[page 24\]](#)

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.

#### [MobiLink Server Shutdown \[page 24\]](#)

The MobiLink server is stopped from the computer where the server was started.

#### [MobiLink Server Logging \[page 25\]](#)

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.

#### [MobiLink Server Use Outside the Current Session \[page 29\]](#)

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.

#### [MobiLink Server in a Server Farm \[page 37\]](#)

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.

#### [Troubleshooting MobiLink Server Startup \[page 38\]](#)

Occasionally you may encounter issues during MobiLink server startup.

## Related Information

[Synchronization Events \[page 332\]](#)

[MobiLink Server System Procedures \[page 582\]](#)

[Adaptive Server Enterprise Consolidated Database \[page 163\]](#)

[IBM DB2 LUW Consolidated Database \[page 165\]](#)  
[Microsoft SQL Server and Microsoft Azure Consolidated Databases \[page 168\]](#)  
[MySQL Consolidated Database \[page 170\]](#)  
[Oracle Consolidated Database \[page 173\]](#)  
[SQL Anywhere Consolidated Database \[page 181\]](#)  
[SAP IQ Consolidated Database \[page 182\]](#)  
[Consolidated Database Setup \[page 155\]](#)  
Introduction to the Relay Server  
Lesson 1: Setting up a MobiLink Consolidated Database  
[MobiLink Server Options \[page 40\]](#)  
[-c mlsrv17 Option \[page 53\]](#)  
[-zs mlsrv17 Option \[page 108\]](#)  
[-o mlsrv17 Option \[page 70\]](#)  
[-v mlsrv17 Option \[page 92\]](#)  
[-x mlsrv17 Option \[page 99\]](#)  
[-c mlsrv17 Option \[page 53\]](#)

## 1.2.1 Required Privileges for MobiLink Server

You must specify a database user for the MobiLink server to connect to the database server. You specify the database user with the mlsrv17 -c option or in the ODBC data source.

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.

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 consolidated database documentation for information about specific consolidated databases.

## Related Information

[Adaptive Server Enterprise Consolidated Database \[page 163\]](#)  
[IBM DB2 LUW Consolidated Database \[page 165\]](#)  
[Microsoft SQL Server and Microsoft Azure Consolidated Databases \[page 168\]](#)  
[MySQL Consolidated Database \[page 170\]](#)  
[Oracle Consolidated Database \[page 173\]](#)  
[SQL Anywhere Consolidated Database \[page 181\]](#)  
[SAP IQ Consolidated Database \[page 182\]](#)

[Consolidated Database Setup \[page 155\]](#)

[-c mlsrv17 Option \[page 53\]](#)

## 1.2.2 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:

```
mlsrv17 ... -x http(port=8080)
```

and the computer is `m11.mycorp.com`, then you can open a web browser and point it to `http://m11.mycorp.com:8080`.

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

### Related Information

[-x mlsrv17 Option \[page 99\]](#)

## 1.2.3 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 Microsoft Windows, right-click the MobiLink server icon in the system tray and click *Shut down*.
- When running on UNIX or Linux without the MobiLink server messages window, type `Q`.
- Use the shutdown method in the MobiLink server API.

### Related Information

[MobiLink Stop Utility \(`mlstop`\) \[page 648\]](#)



## 1.2.4 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`.

```
mlsrv17 -o mlsrv.log -c ...
```

You can control the size of message 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 message log file should be used.
- Use the `-ot` option to specify that a message 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 message 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 message log files when they are assigned a new name based on the date and a sequential number.

### 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.

### Controlling Which Warning Messages Are Reported

You can control which warning messages are reported using the `mlsrv17 -zw`, `-zwd`, and `-zwe` options.

#### In this section:

##### [MobiLink Server Log Viewing \[page 26\]](#)

You can view MobiLink logs in several ways.

##### [MobiLink Server Logging and SAP Passports \[page 26\]](#)

The MobiLink server supports the use of SAP Passports to trace requests from the client through to the backend server.

## Related Information

- [-o mlsrv17 Option \[page 70\]](#)
- [-on mlsrv17 Option \[page 71\]](#)
- [-os mlsrv17 Option \[page 73\]](#)
- [-ot mlsrv17 Option \[page 74\]](#)
- [-zw mlsrv17 Option \[page 111\]](#)
- [-zwd mlsrv17 Option \[page 112\]](#)
- [-zwe mlsrv17 Option \[page 113\]](#)
- [-v mlsrv17 Option \[page 92\]](#)

### 1.2.4.1 MobiLink Server Log Viewing

You can view MobiLink logs in several ways.

- In the MobiLink server messages window
- By opening the message log file
- Using the *MobiLink Server Log File Viewer* in SQL Central

To view log information outside the MobiLink server messages window, you must log the information to a file.

#### *MobiLink Server Log File Viewer*

To view MobiLink server logs, open SQL Central and click **Tools** > **MobiLink 17** > **MobiLink Server Log File Viewer**. You are prompted to choose a server 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 server 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 server log.

### 1.2.4.2 MobiLink Server Logging and SAP Passports

The MobiLink server supports the use of SAP Passports to trace requests from the client through to the backend server.

When a MobiLink client connects using HTTP or HTTPS and the SAP-PASSPORT HTTP header is present, the MobiLink server does the following:

- Extracts and interprets the SAP passport.
- Issues an error if the passport is malformed.



PPK#V3#463500000311ed0aaf5dc15d50c1f75#463500000311ed0aaf5dc15d50bff75#00000000  
000000000000000000000000#0  
I. 2014-02-04 16:11:37. <15> The current synchronization is using a connection  
with connection ID 'SPID 9'#SAP-  
PPK#V3#463500000311ed0aaf5dc15d50c1f75#463500000311ed0aaf5dc15d50bff75#00000000  
000000000000000000000000#0  
I. 2014-02-04 16:11:37. <15> Publication #1: ul\_default\_pub, subscription id: 1,  
last download time: 2014-02-04 16:11:37.790000#SAP-  
PPK#V3#463500000311ed0aaf5dc15d50c1f75#463500000311ed0aaf5dc15d50bff75#00000000  
000000000000000000000000#0  
I. 2014-02-04 16:11:37. <15> The sync sequence ID in the consolidated database:  
041990d3fc7b4af8a8440065ef46de3b; the remote previous sequence ID:  
041990d3fc7b4af8a8440065ef46de3b, and the current sequence ID:  
553e4d716a574ca48c31dc46da1f14db#SAP-  
PPK#V3#463500000311ed0aaf5dc15d50c1f75#463500000311ed0aaf5dc15d50bff75#00000000  
000000000000000000000000#0  
I. 2014-02-04 16:11:37. <15> Log Level: 2#SAP-  
PPK#V3#463500000311ed0aaf5dc15d50c1f75#463500000311ed0aaf5dc15d50bff75#00000000  
000000000000000000000000#0  
I. 2014-02-04 16:11:37. <15> COMMIT Transaction: Begin synchronization#SAP-  
PPK#V3#463500000311ed0aaf5dc15d50c1f75#463500000311ed0aaf5dc15d50bff75#00000000  
000000000000000000000000#0  
I. 2014-02-04 16:11:37. <15> COMMIT Transaction: Upload#SAP-  
PPK#V3#463500000311ed0aaf5dc15d50c1f75#463500000311ed0aaf5dc15d50bff75#00000000  
000000000000000000000000#0  
I. 2014-02-04 16:11:37. <15> COMMIT Transaction: Prepare for download#SAP-  
PPK#V3#463500000311ed0aaf5dc15d50c1f75#463500000311ed0aaf5dc15d50bff75#00000000  
000000000000000000000000#0  
I. 2014-02-04 16:11:37. <15> Next last download timestamp fetched from the  
consolidated database is "2014-02-04 16:11:37.807000"#SAP-  
PPK#V3#463500000311ed0aaf5dc15d50c1f75#463500000311ed0aaf5dc15d50bff75#00000000  
000000000000000000000000#0  
I. 2014-02-04 16:11:37. <15> Insert/Update row [ulhttp02]:#SAP-  
PPK#V3#463500000311ed0aaf5dc15d50c1f75#463500000311ed0aaf5dc15d50bff75#00000000  
000000000000000000000000#0  
I. 2014-02-04 16:11:37. <15> -1#SAP-  
PPK#V3#463500000311ed0aaf5dc15d50c1f75#463500000311ed0aaf5dc15d50bff75#00000000  
000000000000000000000000#0  
I. 2014-02-04 16:11:37. <15> -2#SAP-  
PPK#V3#463500000311ed0aaf5dc15d50c1f75#463500000311ed0aaf5dc15d50bff75#00000000  
000000000000000000000000#0  
I. 2014-02-04 16:11:37. <15> Insert/Update row [ulhttp02]:#SAP-  
PPK#V3#463500000311ed0aaf5dc15d50c1f75#463500000311ed0aaf5dc15d50bff75#00000000  
000000000000000000000000#0  
I. 2014-02-04 16:11:37. <15> -2#SAP-  
PPK#V3#463500000311ed0aaf5dc15d50c1f75#463500000311ed0aaf5dc15d50bff75#00000000  
000000000000000000000000#0  
I. 2014-02-04 16:11:37. <15> -3#SAP-  
PPK#V3#463500000311ed0aaf5dc15d50c1f75#463500000311ed0aaf5dc15d50bff75#00000000  
000000000000000000000000#0  
I. 2014-02-04 16:11:37. <15> Insert/Update row [ulhttp02]:#SAP-  
PPK#V3#463500000311ed0aaf5dc15d50c1f75#463500000311ed0aaf5dc15d50bff75#00000000  
000000000000000000000000#0  
I. 2014-02-04 16:11:37. <15> -3#SAP-  
PPK#V3#463500000311ed0aaf5dc15d50c1f75#463500000311ed0aaf5dc15d50bff75#00000000  
000000000000000000000000#0  
I. 2014-02-04 16:11:37. <15> -4#SAP-  
PPK#V3#463500000311ed0aaf5dc15d50c1f75#463500000311ed0aaf5dc15d50bff75#00000000  
000000000000000000000000#0  
I. 2014-02-04 16:11:37. <15> Sending the download to the remote database#SAP-  
PPK#V3#463500000311ed0aaf5dc15d50c1f75#463500000311ed0aaf5dc15d50bff75#00000000  
000000000000000000000000#0  
I. 2014-02-04 16:11:37. <15> COMMIT Transaction: Download#SAP-  
PPK#V3#463500000311ed0aaf5dc15d50c1f75#463500000311ed0aaf5dc15d50bff75#00000000  
000000000000000000000000#0  
I. 2014-02-04 16:11:37. <15> COMMIT Transaction: End synchronization#SAP-  
PPK#V3#463500000311ed0aaf5dc15d50c1f75#463500000311ed0aaf5dc15d50bff75#00000000  
000000000000000000000000#0

```
I. 2014-02-04 16:11:37. <15> Synchronization complete#SAP-  
PPK#V3#4635000000311ed0aaf5dc15d50c1f75#4635000000311ed0aaf5dc15d50bfff75#00000000  
000000000000000000000000#0
```

## Related Information

[-ncs mlsrv17 Option \[page 67\]](#)

[-ncsd mlsrv17 Option \[page 68\]](#)

[-ncsp mlsrv17 Option \[page 69\]](#)

## 1.2.5 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 `mlsrv17 -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.

### In this section:

#### [Running the MobiLink Server as a Daemon on UNIX/Linux \[page 30\]](#)

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

#### [MobiLink Server as a Service on Microsoft Windows \[page 30\]](#)

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

## Related Information

[How to Run the Database Server as a Service or Daemon](#)

[-ud mlsrv17 Option \[page 90\]](#)

[MobiLink Stop Utility \(mlstop\) \[page 648\]](#)

[Service Utility \(dbsvc\) for Linux](#)

## 1.2.5.1 Running the MobiLink Server as a Daemon on UNIX/Linux

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

### Procedure

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

```
mllsrv17 -c "DSN=SQL Anywhere 17 Demo;UID=DBA;PWD=sql" -ud
```

### Results

The UNIX MobiLink server is running as a daemon.

### Related Information

[-ud mllsrv17 Option \[page 90\]](#)  
[Service Utility \(dbsvc\) for Linux](#)

## 1.2.5.2 MobiLink Server as a Service on Microsoft Windows

To run the Microsoft 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 SQL Central:

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

### In this section:

[Working with Services \[page 31\]](#)

Use SQL 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.

[Startup Options for Services \[page 32\]](#)

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

#### [Command Line Options \[page 33\]](#)

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.

#### [Account Options \[page 33\]](#)

You can choose which account the service runs under. Most services run under the special LocalSystem account, which is the default option for services.

#### [Changing the Executable File \[page 34\]](#)

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

#### [Starting and Stopping a Service \[page 34\]](#)

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

#### [Multiple Services \[page 35\]](#)

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 SQL Central to perform all the service management for MobiLink.

## Related Information

[Service Utility \(dbsvc\) for Windows](#)

### 1.2.5.2.1 Working with Services

Use SQL 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.

## Context

The service icons in SQL 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.

## Procedure

1. In SQL Central, in the left pane, click MobiLink 17.

2. In the right pane, click the [Services](#) tab.
3. Perform one of the following tasks:
  - To create a new service, click ► [File](#) ► [New](#) ► [Service](#) ► and then follow the instructions in the [Create Service Wizard](#).
  - To delete a service, choose the service and then click ► [Edit](#) ► [Delete](#) ►. Click [Yes](#) to confirm the deletion of the service.
  - To modify an existing service, choose the service and then click ► [File](#) ► [Properties](#) ► and then edit the service properties.

## Results

The service is added, deleted or modified.

## Related Information

[Service Utility \(dbsvc\) for Windows](#)

### 1.2.5.2.2 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.



## 1.2.5.2.3 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 17 Demo;UID=DBA;PWD=sql"  
-vc
```

The command line options for a service are the same as those for the executable.

## Related Information

[MobiLink Server Options \[page 40\]](#)

## 1.2.5.2.4 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 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.

## 1.2.5.2.5 Changing the Executable File

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

### Context

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

### Procedure

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.

## 1.2.5.2.6 Starting and Stopping a Service

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

### Context

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

### Procedure

1. In SQL Central, click MobiLink 17 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.

### 1.2.5.2.7 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 SQL 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 might 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 SQL Central.

## Service Groups

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

### In this section:

#### [Checking and Changing Which Group a Service Belongs to \[page 36\]](#)

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 SQL Central.

#### [Adding a Service or Group to a List of Dependencies \[page 37\]](#)

With SQL Central, you can specify dependencies for a service. For example, you can ensure that at least one group has started before the current service.

## 1.2.5.2.7.1 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 SQL Central.

### Procedure

1. In SQL Central, click MobiLink 17 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

The service is assigned to the given group.

### Next Steps

You can configure your services to ensure they start in the correct order.

### Related Information

[Adding a Service or Group to a List of Dependencies \[page 37\]](#)

## 1.2.5.2.7.2 Adding a Service or Group to a List of Dependencies

With SQL Central, you can specify dependencies for a service. For example, you can ensure that at least one group has started before the current service.

### Procedure

1. In SQL Central, click MobiLink 17 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.

## 1.2.6 MobiLink Server in a Server Farm

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 MobiLink servers, restartable download is supported, but you must configure the load balancer in front of the MobiLink servers to ensure that all requests with the same value in the ml-client-id HTTP header go to the same MobiLink server.

For HTTP-based streams, one synchronization can require multiple HTTP requests and so you must either use the Relay Server, or configure session-based affinity in your load balancer to ensure requests all go to same MobiLink server. The MobiLink server sets two HTTP headers which can be used to control server affinity: the ml-session-id header is a UUID that is unique for each synchronization; the ml-client-id header is unique for each remote database. If your load balancer cannot be configured to use either of those headers, you can use the session\_id\_cookie or client\_id\_cookie stream options to cause the server to also set a cookie with the given name containing the ml-session-id or ml-client-id value, respectively. For example

```
-x http(port=8081;session_id_cookie=JSESSIONID)
```

The above will set the commonly used JSESSIONID cookie with the value of the ml-session-id header.

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`:

```
m1srv17 -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.

## Related Information

[MobiLink Arbiter Server Utility for Windows \(mlarbiter\) \[page 658\]](#)

[-lsc m1srv17 Option \[page 65\]](#)

[-ca m1srv17 Option \[page 54\]](#)

[-x m1srv17 Option \[page 99\]](#)

## 1.2.7 Troubleshooting MobiLink Server Startup

Occasionally you may encounter issues during MobiLink server startup.

### In this section:

[Ensure That Network Communication Software is Running \[page 39\]](#)

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. Confirm that other software requiring network communications is working properly before running the MobiLink server.

[Debug Network Communications Startup Problems \[page 39\]](#)

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.

[Verify MobiLink Server Connectivity \[page 39\]](#)

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.

## 1.2.7.1 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. 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.

## 1.2.7.2 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.

### Related Information

[MobiLink Server Logging \[page 25\]](#)

## 1.2.7.3 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:

```
m1srv17 ... -x http(port=8080)
```

and the computer is `m11.mycorp.com`, then you can open a web browser and point it to `http://m11.mycorp.com:8080`.

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

### Related Information

[Introduction to the Relay Server](#)

## 1.3 MobiLink Server Options

The following options are available for use with the MobiLink server.

### In this section:

#### [mlsrv17 Syntax \[page 44\]](#)

The mlsrv17 command is used to start a MobiLink server.

#### [@data mlsrv17 Option \[page 50\]](#)

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

#### [-a mlsrv17 Option \[page 51\]](#)

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

#### [-b mlsrv17 Option \[page 51\]](#)

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

#### [-bn mlsrv17 Option \[page 52\]](#)

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

#### [-c mlsrv17 Option \[page 53\]](#)

Specifies connection parameters for the consolidated database.

#### [-ca mlsrv17 Option \[page 54\]](#)

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

#### [-cinit mlsrv17 Option \[page 54\]](#)

Sets the initial size for the server memory cache.

#### [-cn mlsrv17 Option \[page 55\]](#)

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

#### [-cr mlsrv17 Option \[page 56\]](#)

Sets the maximum number of database connection retries.

#### [-cs mlsrv17 Option \[page 56\]](#)

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

#### [-ct mlsrv17 Option \[page 57\]](#)

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

#### [-dl mlsrv17 Option \[page 57\]](#)

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

#### [-dr mlsrv17 Option \[page 58\]](#)

For Adaptive Server Enterprise only. This can be used if all tables involved in synchronization do not use the datarows locking scheme.

#### [-ds mlsrv17 Option \[page 58\]](#)

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



- dsd mlsrv17 Option [page 59]  
Disables snapshot isolation.
- dt mlsrv17 Option [page 60]  
For Microsoft SQL Server, Microsoft Azure, SAP Adaptive Server Enterprise, and Oracle databases only.  
Causes MobiLink to detect transactions only within the current database.
- e mlsrv17 Option [page 61]  
Stores error logs sent from SQL Anywhere MobiLink clients.
- esu mlsrv17 Option [page 62]  
Use snapshot isolation for uploads.
- et mlsrv17 Option [page 63]  
Stores error logs sent from SQL Anywhere MobiLink clients in the named file after truncating the existing file.
- fips mlsrv17 Option [page 63]  
Forces all secure MobiLink streams to use FIPS-certified modules.
- ftr mlsrv17 Option [page 64]  
Specifies a location for files that are to be downloaded by the mlfiletransfer utility or by the MobiLink Agent.
- ftru mlsrv17 Option [page 65]  
Specifies a location for files that are to be uploaded with the mlfiletransfer utility or by the MobiLink Agent.
- lsc mlsrv17 Option [page 65]  
Specifies the connection information for the local server. This information is passed to other servers in the server farm.
- nc mlsrv17 Option [page 66]  
Sets the maximum number of concurrent network connections.
- ncs mlsrv17 Option [page 67]  
Reads the first NCS (Native Component Supportability) configuration file in the path or current directory. The configuration file contains the settings required to connect to an SAP Diagnostic Agent.
- ncsd mlsrv17 Option [page 68]  
Reads the NCS (Native Component Supportability) configuration file in the specified directory. The configuration file contains the settings required to connect to an SAP Diagnostic Agent.
- ncsp mlsrv17 Option [page 69]  
Specifies **name=value** pairs to configure the NCS (Native Component Supportability) library.
- notifier mlsrv17 Option [page 69]  
Starts the Notifier for server-initiated synchronization.
- o mlsrv17 Option [page 70]  
Logs output messages to a MobiLink server message log file, and limits the data logged to the MobiLink server messages window.
- on mlsrv17 Option [page 71]  
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.
- oq mlsrv17 Option [page 72]  
On Windows, prevents the appearance of the error window when a startup error occurs.

- [-os mlsrv17 Option \[page 73\]](#)  
Sets the maximum size of current and old MobiLink server message log files.
- [-ot mlsrv17 Option \[page 74\]](#)  
Logs output messages to the MobiLink server message log file, but deletes the contents first.
- [-ppv mlsrv17 Option \[page 74\]](#)  
Causes MobiLink to print new periodic monitoring values according to the period specified. Periods are in seconds.
- [-q mlsrv17 Option \[page 79\]](#)  
Instructs MobiLink to run with a minimized messages window on startup.
- [-r mlsrv17 Option \[page 79\]](#)  
Sets the maximum number of deadlock retries.
- [-rd mlsrv17 Option \[page 79\]](#)  
Sets the maximum delay time between deadlock retries.
- [-rp mlsrv17 Option \[page 80\]](#)  
Specifies the directory to which synchronizations are recorded for playback with the mlreplay utility.
- [-rrp mlsrv17 Option \[page 81\]](#)  
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.
- [-s mlsrv17 Option \[page 81\]](#)  
Sets the maximum number of rows that can be uploaded at the same time.
- [-sl dnet mlsrv17 Option \[page 82\]](#)  
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.
- [-sl java mlsrv17 Option \[page 84\]](#)  
Sets the Java VM options and forces the Java VM to load on startup. This option is recommended when using Java scripting logic.
- [-sm mlsrv17 Option \[page 85\]](#)  
Sets the maximum number of synchronizations that can be actively worked on by limiting the maximum number of network connections.
- [-tc mlsrv17 Option \[page 86\]](#)  
Sets a timeout threshold for long running SQL scripts.
- [-tf mlsrv17 Option \[page 87\]](#)  
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.
- [-ts mlsrv17 Option \[page 88\]](#)  
Sets up a MobiLink server tracing session.
- [-tx mlsrv17 Option \[page 89\]](#)  
When using transactional uploads, this option batches groups of transactions and commits them together.
- [-ud mlsrv17 Option \[page 90\]](#)  
Instructs MobiLink to run as a daemon.
- [-ui mlsrv17 Option \[page 91\]](#)

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

[-ux mlsrv17 Option \[page 91\]](#)

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

[-v mlsrv17 Option \[page 92\]](#)

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

[-w mlsrv17 Option \[page 96\]](#)

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

[-wm mlsrv17 Option \[page 97\]](#)

Sets the maximum number of concurrent database worker threads.

[-wn mlsrv17 Option \[page 98\]](#)

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

[-wu mlsrv17 Option \[page 98\]](#)

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

[-x mlsrv17 Option \[page 99\]](#)

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

[-zf mlsrv17 Option \[page 107\]](#)

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

[-zp mlsrv17 Option \[page 108\]](#)

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

[-zs mlsrv17 Option \[page 108\]](#)

Specifies a MobiLink server name for mlstop.

[-zt mlsrv17 Option \[page 109\]](#)

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

[-zu mlsrv17 Option \[page 109\]](#)

Controls the automatic addition of users when the authenticate\_user and authenticate\_user\_hashed scripts are undefined.

[-zup mlsrv17 Option \[page 110\]](#)

Specifies the default user authentication policy to be used to authenticate a user against the LDAP server.

[-zus mlsrv17 Option \[page 111\]](#)

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

[-zw mlsrv17 Option \[page 111\]](#)

Controls which levels of warning message to display.

[-zwd mlsrv17 Option \[page 112\]](#)

Disables specific warning codes.

[-zwe mlsrv17 Option \[page 113\]](#)

Enables specific warning codes.

## 1.3.1 mlsrv17 Syntax

The mlsrv17 command is used to start a MobiLink server.

### ≡ Syntax

```
mlsrv17 -c "connection-string" [ options ]
```

Option	Description
@ data	Read in options from the specified environment variable or configuration file.
-a	Keep using a consolidated database connection after a synchronization error on that connection.
-b	Trim blank padding of strings.
-bn size	Specify the maximum number of bytes to consider when comparing BLOBs for conflict detection.
-c "keyword=value;..."	Supply ODBC database connection parameters for your consolidated database. This option is required.
-ca host_or_ip	Set the host name or IP address for the MobiLink arbiter server.
-cinit size	Specify the initial size for the server memory cache.
-cn connections	Set the maximum number of simultaneous connections with the consolidated database server.
-cr count	Set the maximum number of database connection retries.
-CS "keyword=value;..."	Supply system database connection parameters for your MobiLink System Database (MLSD).
-ct connection-timeout	Set the length of time a connection may be unused before it is timed out.
-dl	Display all log messages in the MobiLink server messages window.
-dr	For Adaptive Server Enterprise only. Ensures that tables involved in synchronization do not use the DataRow locking scheme.
-ds size	Specify the maximum amount of data that can be stored for use in all restartable downloads.
-dsd	Disable snapshot isolation, which is the default download isolation level for SQL Anywhere and Microsoft SQL Server consolidated databases.
-dt	Detect transactions only within the current database.
-e filename	Store remote database error logs sent into the named file.
-esu	Use snapshot isolation for uploads.
-et filename	Store remote database error logs sent into the named file, but delete the contents first if the file exists.

Option	Description
<code>-fips</code>	Forces all secure MobiLink streams to be FIPS-certified.
<code>-ftr path</code>	Specifies a location for files that are to be downloaded by the MobiLink transfer utility (mlfiletransfer).
<code>-ftru</code>	Specifies a location for files uploaded with the MobiLink File Transfer utility (mlfiletransfer).
<code>-lsc protocol[protocol-options]</code>	Specifies the local server connect information.
<code>-nc connections</code>	Sets maximum number of concurrent network connections.
<code>-ncs</code>	Reads the first NCS (Native Component Supportability) configuration file in the path or current directory.
<code>-ncsd directory</code>	Reads the NCS (Native Component Supportability) configuration file in the specified directory.
<code>-ncsp name=value;...</code>	Configures the NCS library using name=value pairs.
<code>-notifier</code>	Starts a Notifier for server-initiated synchronization.
<code>-o logfile</code>	Log messages to a file.
<code>-on size</code>	Set maximum size for log file.
<code>-oq</code>	Prevent the popup window on startup error.
<code>-os size</code>	Maximum size of old log files.
<code>-ot logfile</code>	Log messages to a file, but delete its contents first.
<code>-ppv period</code>	Causes MobiLink to print new periodic monitoring values according to the period specified.
<code>-q</code>	Minimize the MobiLink server messages window.
<code>-r retries</code>	Retry deadlocked uploads at most this many times.
<code>-rd delay</code>	Set maximum delay, in seconds, before retrying a deadlocked transaction.
<code>-rp directory</code>	Specifies the directory to which synchronizations are recorded for playback with the mlreplay utility.
<code>-rrp directory</code>	Causes the MobiLink server to run the mlreplay utility in the given directory when the server starts.
<code>-s count</code>	Specify the maximum number of rows to be uploaded to or fetched from the consolidated database at once.
<code>-sl dnet script-options</code>	Set the .NET Common Language Runtime (CLR) options and force the CLR to load on startup.
<code>-sl java script-options</code>	Set the Java VM options and force loading of the Java VM on startup.
<code>-sm number</code>	Set the maximum number of synchronizations that can be worked on concurrently.
<code>-tc minutes</code>	Set the timeout threshold for SQL script execution.

Option	Description
<code>-tf</code>	Fail the SQL script execution when the timeout threshold is reached (not for Oracle).
<code>-ts session-name(session-option=option-value[:...])</code>	Sets up a MobiLink server tracing session.
<code>-tx count</code>	For transactional uploads, batches groups of transactions and commits them together.
<code>-ud</code>	On UNIX and Linux platforms, run as a daemon.
<code>-ui</code>	For Linux with X windows, starts the MobiLink server in shell mode if a usable display isn't available.
<code>-ux</code>	For Linux with X windows, opens the MobiLink server messages window.
<code>-v [ levels ]</code>	Controls the type of messages written to the message log file.
<code>-W count</code>	Set the initial number of database worker threads.
<code>-wm count</code>	Set the maximum number of database worker threads.
<code>-wn count</code>	Set the number of network worker threads for concurrent processing of network streams.
<code>-wu count</code>	Set the maximum number of database worker threads permitted to process uploads concurrently.
<code>-X protocol "options;..."</code>	Specify the communications protocol. Optionally, specify network parameters in form <code>parameter=value</code> , with multiple parameters separated by semicolons.
<code>-zf</code>	Specifies that the MobiLink server should check for script changes at the beginning of each synchronization.
<code>-zp</code>	Ignore some apparent differences between <code>TIMESTAMP</code> values when the remote and consolidated databases have different precision.
<code>-ZS name</code>	Specify a server name.
<code>-zt number</code>	Specify the maximum number of processors used to run the MobiLink server.
<code>-zu { +   - }</code>	Controls the automatic addition of users when the <code>authenticate_user</code> script is undefined.
<code>-zup name</code>	Specifies the default user authentication policy to be used to authenticate a user against the LDAP server.
<code>-zus</code>	Causes MobiLink to invoke upload scripts for tables for which there is no upload.
<code>-zw 1,...5</code>	Controls which levels of warning message to display.
<code>-zwd code</code>	Disables specific warning codes.
<code>-zwe code</code>	Enables specific warning codes.

## 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.

## Related Information

[MobiLink Server \[page 21\]](#)  
[@data mlsrv17 Option \[page 50\]](#)  
[-a mlsrv17 Option \[page 51\]](#)  
[-b mlsrv17 Option \[page 51\]](#)  
[-bn mlsrv17 Option \[page 52\]](#)  
[-c mlsrv17 Option \[page 53\]](#)  
[-ca mlsrv17 Option \[page 54\]](#)  
[-cinit mlsrv17 Option \[page 54\]](#)  
[-cn mlsrv17 Option \[page 55\]](#)  
[-cr mlsrv17 Option \[page 56\]](#)  
[-cs mlsrv17 Option \[page 56\]](#)  
[-ct mlsrv17 Option \[page 57\]](#)  
[-dl mlsrv17 Option \[page 57\]](#)  
[-dr mlsrv17 Option \[page 58\]](#)  
[-ds mlsrv17 Option \[page 58\]](#)  
[-dsd mlsrv17 Option \[page 59\]](#)  
[-dt mlsrv17 Option \[page 60\]](#)  
[-e mlsrv17 Option \[page 61\]](#)  
[-esu mlsrv17 Option \[page 62\]](#)  
[-et mlsrv17 Option \[page 63\]](#)  
[-fips mlsrv17 Option \[page 63\]](#)  
[-ftr mlsrv17 Option \[page 64\]](#)  
[-ftru mlsrv17 Option \[page 65\]](#)  
[-lsc mlsrv17 Option \[page 65\]](#)  
[-nc mlsrv17 Option \[page 66\]](#)  
[-ncs mlsrv17 Option \[page 67\]](#)  
[-ncsd mlsrv17 Option \[page 68\]](#)  
[-ncsp mlsrv17 Option \[page 69\]](#)  
[-notifier mlsrv17 Option \[page 69\]](#)  
[-o mlsrv17 Option \[page 70\]](#)

[-on mlsrv17 Option \[page 71\]](#)  
[-oq mlsrv17 Option \[page 72\]](#)  
[-os mlsrv17 Option \[page 73\]](#)  
[-ot mlsrv17 Option \[page 74\]](#)  
[-ppv mlsrv17 Option \[page 74\]](#)  
[-q mlsrv17 Option \[page 79\]](#)  
[-r mlsrv17 Option \[page 79\]](#)  
[-rd mlsrv17 Option \[page 79\]](#)  
[-rp mlsrv17 Option \[page 80\]](#)  
[-rrp mlsrv17 Option \[page 81\]](#)  
[-s mlsrv17 Option \[page 81\]](#)  
[-sl dnet mlsrv17 Option \[page 82\]](#)  
[-sl java mlsrv17 Option \[page 84\]](#)  
[-sm mlsrv17 Option \[page 85\]](#)  
[-tc mlsrv17 Option \[page 86\]](#)  
[-tf mlsrv17 Option \[page 87\]](#)  
[-ts mlsrv17 Option \[page 88\]](#)  
[-tx mlsrv17 Option \[page 89\]](#)  
[-ud mlsrv17 Option \[page 90\]](#)  
[-ui mlsrv17 Option \[page 91\]](#)  
[-ux mlsrv17 Option \[page 91\]](#)  
[-v mlsrv17 Option \[page 92\]](#)  
[-w mlsrv17 Option \[page 96\]](#)  
[-wm mlsrv17 Option \[page 97\]](#)  
[-wn mlsrv17 Option \[page 98\]](#)  
[-wu mlsrv17 Option \[page 98\]](#)  
[-x mlsrv17 Option \[page 99\]](#)  
[-zf mlsrv17 Option \[page 107\]](#)  
[-zp mlsrv17 Option \[page 108\]](#)  
[-zs mlsrv17 Option \[page 108\]](#)  
[-zt mlsrv17 Option \[page 109\]](#)  
[-zu mlsrv17 Option \[page 109\]](#)  
[-zus mlsrv17 Option \[page 111\]](#)  
[-zw mlsrv17 Option \[page 111\]](#)  
[-zwd mlsrv17 Option \[page 112\]](#)  
[-zwe mlsrv17 Option \[page 113\]](#)  
[MobiLink Server \[page 21\]](#)  
[@data mlsrv17 Option \[page 50\]](#)  
[-a mlsrv17 Option \[page 51\]](#)  
[-b mlsrv17 Option \[page 51\]](#)  
[-bn mlsrv17 Option \[page 52\]](#)  
[-c mlsrv17 Option \[page 53\]](#)  
[-ca mlsrv17 Option \[page 54\]](#)



-cinit mlsrv17 Option [page 54]  
-cn mlsrv17 Option [page 55]  
-cr mlsrv17 Option [page 56]  
-cs mlsrv17 Option [page 56]  
-ct mlsrv17 Option [page 57]  
-dl mlsrv17 Option [page 57]  
-dr mlsrv17 Option [page 58]  
-ds mlsrv17 Option [page 58]  
-dsd mlsrv17 Option [page 59]  
-dt mlsrv17 Option [page 60]  
-e mlsrv17 Option [page 61]  
-esu mlsrv17 Option [page 62]  
-et mlsrv17 Option [page 63]  
-fips mlsrv17 Option [page 63]  
-ftr mlsrv17 Option [page 64]  
-ftru mlsrv17 Option [page 65]  
-lsc mlsrv17 Option [page 65]  
-nc mlsrv17 Option [page 66]  
-notifier mlsrv17 Option [page 69]  
-o mlsrv17 Option [page 70]  
-on mlsrv17 Option [page 71]  
-oq mlsrv17 Option [page 72]  
-os mlsrv17 Option [page 73]  
-ot mlsrv17 Option [page 74]  
-ppv mlsrv17 Option [page 74]  
-q mlsrv17 Option [page 79]  
-r mlsrv17 Option [page 79]  
-rd mlsrv17 Option [page 79]  
-rp mlsrv17 Option [page 80]  
-rrp mlsrv17 Option [page 81]  
-s mlsrv17 Option [page 81]  
-sl dnet mlsrv17 Option [page 82]  
-sl java mlsrv17 Option [page 84]  
-sm mlsrv17 Option [page 85]  
-tc mlsrv17 Option [page 86]  
-tf mlsrv17 Option [page 87]  
-ts mlsrv17 Option [page 88]  
-tx mlsrv17 Option [page 89]  
-ud mlsrv17 Option [page 90]  
-ui mlsrv17 Option [page 91]  
-ux mlsrv17 Option [page 91]  
-v mlsrv17 Option [page 92]  
-w mlsrv17 Option [page 96]

[-wm mlsrv17 Option \[page 97\]](#)  
[-wn mlsrv17 Option \[page 98\]](#)  
[-wu mlsrv17 Option \[page 98\]](#)  
[-x mlsrv17 Option \[page 99\]](#)  
[-zf mlsrv17 Option \[page 107\]](#)  
[-zp mlsrv17 Option \[page 108\]](#)  
[-zs mlsrv17 Option \[page 108\]](#)  
[-zt mlsrv17 Option \[page 109\]](#)  
[-zu mlsrv17 Option \[page 109\]](#)  
[-zus mlsrv17 Option \[page 111\]](#)  
[-zw mlsrv17 Option \[page 111\]](#)  
[-zwd mlsrv17 Option \[page 112\]](#)  
[-zwe mlsrv17 Option \[page 113\]](#)

## 1.3.2 @data mlsrv17 Option

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

### ☰ Syntax

```
mlsrv17 -c "connection-string" @data ...
```

### Remarks

Use this option to read mlsrv17 command line options from the specified environment variable or configuration file. If both exist with the name that is specified, the environment variable is used.

To protect information in the configuration file, you can use the File Hiding utility (dbfhide) to encode the contents of the configuration file.

### Related Information

[Configuration Files](#)

[File Hiding Utility \(dbfhide\)](#)

[Configuration Files](#)

[File Hiding Utility \(dbfhide\)](#)

### 1.3.3 -a mlsrv17 Option

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

#### ≡ Syntax

```
mlsrv17 -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.

### 1.3.4 -b mlsrv17 Option

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

#### ≡ Syntax

```
mlsrv17 -c "connection-string" -b ...
```

#### Remarks

##### i Note

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.

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.

## Related Information

[RDBMS-Dependent Synchronization Scripts \[page 157\]](#)

[NVARCHAR Data Type](#)

[upload\\_fetch Table Event \[page 502\]](#)

[upload\\_fetch\\_column\\_conflict Table Event \[page 504\]](#)

[RDBMS-Dependent Synchronization Scripts \[page 157\]](#)

[NCHAR Data Type](#)

[upload\\_fetch Table Event \[page 502\]](#)

[upload\\_fetch\\_column\\_conflict Table Event \[page 504\]](#)

### 1.3.5 -bn mlsrv17 Option

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

☞ Syntax

```
mlsrv17 -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.

## 1.3.6 -c mlsrv17 Option

Specifies connection parameters for the consolidated database.

### ≡ Syntax

```
mlsrv17 -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.

You can use the File Hiding utility (`dbfhide`) to hide the password.

## Example

```
mlsrv17 -c "DSN=odbcname;UID=DBA;PWD=passwd"
```

## Related Information

[Alphabetical List of Connection Parameters](#)

[File Hiding Utility \(dbfhide\)](#)

[Alphabetical List of Connection Parameters](#)

[File Hiding Utility \(dbfhide\)](#)

## 1.3.7 -ca mlsrv17 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

```
mlsrv17 -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.

### i 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.

### Related Information

[-lsc mlsrv17 Option \[page 65\]](#)

[-notifier mlsrv17 Option \[page 69\]](#)

[-lsc mlsrv17 Option \[page 65\]](#)

[-notifier mlsrv17 Option \[page 69\]](#)

## 1.3.8 -cinit mlsrv17 Option

Sets the initial size for the server memory cache.

### ≡ Syntax

```
mlsrv17 -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.

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`.

## 1.3.9 -cn mlsrv17 Option

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

### Syntax

```
mlsrv17 -c "connection-string" -cn value ...
```

## Remarks

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

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

A value larger than the number of database worker threads may speed performance, particularly if connecting to the consolidated database is slow or if multiple script versions are in use. The optimum maximum number of database connections is the number of script versions times the number of database worker threads.

Connections above this optimum value do not necessarily speed synchronization, and needlessly consume resources in both the MobiLink server and the consolidated database server.

## Related Information

[-w mlsrv17 Option \[page 96\]](#)

[-w mlsrv17 Option \[page 96\]](#)

### 1.3.10 -cr mlsrv17 Option

Sets the maximum number of database connection retries.

#### ☰ Syntax

```
mlsrv17 -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.

### 1.3.11 -cs mlsrv17 Option

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

#### ☰ Syntax

```
mlsrv17 -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.

## 1.3.12 -ct mlsrv17 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

```
mlsrv17 -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.

## 1.3.13 -dl mlsrv17 Option

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

### Syntax

```
mlsrv17 -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.

## Related Information

[How to Log Database Server Messages to a File](#)

[-o mlsrv17 Option \[page 70\]](#)

[How to Log Database Server Messages to a File](#)

[-o mlsrv17 Option \[page 70\]](#)

### 1.3.14 -dr mlsrv17 Option

For Adaptive Server Enterprise only. This can be used if all tables involved in synchronization do not use the datarows locking scheme.

#### ≡ Syntax

```
mlsrv17 -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.

## Related Information

[MobiLink Isolation Levels \[page 149\]](#)

[MobiLink Isolation Levels \[page 149\]](#)

### 1.3.15 -ds mlsrv17 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

```
mlsrv17 -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.

## Related Information

[Resumption of Failed Downloads \[page 144\]](#)

[-dc dbmlsync Option](#)

[Resume Partial Download Synchronization Parameter](#)

[Resumption of Failed Downloads \[page 144\]](#)

[-dc dbmlsync Option](#)

[Resume Partial Download Synchronization Parameter](#)

### 1.3.16 -dsd mlsrv17 Option

Disables snapshot isolation.

☰ Syntax

```
mlsrv17 -c "connection-string" -dsd ...
```

## Remarks

When the consolidated database is SQL Anywhere or Microsoft SQL Server, the default isolation level for downloads is snapshot isolation.

You can also change the default isolation level in a script. However, for SQL Anywhere and Microsoft SQL Server 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, then it may be overridden in the `begin_upload` and `begin_download` scripts.

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

In Microsoft Azure consolidated databases, by default, uncommitted operations (inserts, updates, and deletes) do not prevent other connections from accessing the same tables. Because of this behavior, do not use the `-dsd` option (disable snapshot isolation for download) when you use timestamp-based download logic for your synchronization. Doing so can cause data inconsistencies.

## Related Information

[MobiLink Isolation Levels \[page 149\]](#)

[-dt mlsrv17 Option \[page 60\]](#)

[-esu mlsrv17 Option \[page 62\]](#)

[MobiLink Isolation Levels \[page 149\]](#)

[-dt mlsrv17 Option \[page 60\]](#)

[-esu mlsrv17 Option \[page 62\]](#)

## 1.3.17 -dt mlsrv17 Option

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

### ≡ Syntax

```
mlsrv17 -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, Microsoft Azure, SAP Adaptive Server Enterprise, or Oracle that is also running other databases.
- You are using snapshot isolation for uploads or downloads with Microsoft SQL Server or Microsoft Azure.
- 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 or Microsoft Azure databases using snapshot isolation, and Adaptive Server Enterprise databases using the DataRow locking scheme for tables involved in synchronization.

For Oracle consolidated databases, the MobiLink user that connects to the Oracle database must have select permission on the following global view in the Oracle database: GV\$TRANSACTION, GV\$SESSION, and GV\$LOCKED\_OBJECT.

## Related Information

[MobiLink Isolation Levels \[page 149\]](#)

[-dsd mlsrv17 Option \[page 59\]](#)

[-esu mlsrv17 Option \[page 62\]](#)

[MobiLink Isolation Levels \[page 149\]](#)

[-dsd mlsrv17 Option \[page 59\]](#)

[-esu mlsrv17 Option \[page 62\]](#)

## 1.3.18 -e mlsrv17 Option

Stores error logs sent from SQL Anywhere MobiLink clients.

☰ Syntax

```
mlsrv17 -c "connection-string" -e filename ...
```

## Remarks

With no `-e` option, error logs from SQL Anywhere MobiLink clients are stored in a file named `mlsrv17.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`.

## Related Information

[-et mlsrv17 Option \[page 63\]](#)

[ErrorLogSendLimit \(el\) Extended Option](#)

[-et mlsrv17 Option \[page 63\]](#)

[ErrorLogSendLimit \(el\) Extended Option](#)

## 1.3.19 -esu mlsrv17 Option

Use snapshot isolation for uploads.

### ≡ Syntax

```
mlsrv17 -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, Microsoft SQL Server, and Microsoft Azure consolidated databases.

## Related Information

[MobiLink Isolation Levels \[page 149\]](#)

[-dsd mlsrv17 Option \[page 59\]](#)

[-dt mlsrv17 Option \[page 60\]](#)

[-r mlsrv17 Option \[page 79\]](#)

[-rd mlsrv17 Option \[page 79\]](#)

[MobiLink Isolation Levels \[page 149\]](#)

[-dsd mlsrv17 Option \[page 59\]](#)

[-dt mlsrv17 Option \[page 60\]](#)

[-r mlsrv17 Option \[page 79\]](#)

[-rd mlsrv17 Option \[page 79\]](#)

## 1.3.20 -et mlsrv17 Option

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

### ≡ Syntax

```
mlsrv17 -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.

### Related Information

[ErrorLogSendLimit \(el\) Extended Option](#)

[-e mlsrv17 Option \[page 61\]](#)

[ErrorLogSendLimit \(el\) Extended Option](#)

[-e mlsrv17 Option \[page 61\]](#)

## 1.3.21 -fips mlsrv17 Option

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

### ≡ Syntax

```
mlsrv17 -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 connections that employ simple obfuscation.

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`.

For MobiLink transport layer security, the `-fips` option causes the server to use the FIPS-certified RSA encryption algorithm, even the `-x` option does not contain a `fips` setting.

FIPS-certified encryption requires a separate license.

## Related Information

[MobiLink Client/Server Communications Encryption \[page 196\]](#)

[FIPS-certified Encryption Technology](#)

[Separately Licensed Components](#)

[MobiLink Client/Server Communications Encryption \[page 196\]](#)

[FIPS-certified Encryption Technology](#)

[Separately Licensed Components](#)

### 1.3.22 `-ftr mlsrv17` Option

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

#### ≡ Syntax

```
mlsrv17 -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 `mfiletransfer` utility to download files.

## Related Information

[MobiLink File Transfer Utility \(mfiletransfer\)](#)

[-ftru mlsrv17 Option \[page 65\]](#)

[authenticate\\_file\\_transfer Connection Event \[page 348\]](#)



[MobiLink File Transfer Utility \(mlfiletransfer\)](#)

[-ftru mlsrv17 Option \[page 65\]](#)

[authenticate\\_file\\_transfer Connection Event \[page 348\]](#)

### 1.3.23 -ftru mlsrv17 Option

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

#### ☰ Syntax

```
mlsrv17 -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.

#### Related Information

[MobiLink File Transfer Utility \(mlfiletransfer\)](#)

[-ftr mlsrv17 Option \[page 64\]](#)

[authenticate\\_file\\_transfer Connection Event \[page 348\]](#)

[MobiLink File Transfer Utility \(mlfiletransfer\)](#)

[-ftr mlsrv17 Option \[page 64\]](#)

[authenticate\\_file\\_transfer Connection Event \[page 348\]](#)

### 1.3.24 -lsc mlsrv17 Option

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

#### ☰ Syntax

```
mlsrv17 -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:

```
mlsrv17 -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.

## Related Information

[MobiLink Server in a Server Farm \[page 37\]](#)

[Notifiers in a MobiLink Server Farm](#)

[-rrp mlsrv17 Option \[page 81\]](#)

[-zs mlsrv17 Option \[page 108\]](#)

[-rrp mlsrv17 Option \[page 81\]](#)

[MobiLink Server in a Server Farm \[page 37\]](#)

[Notifiers in a MobiLink Server Farm](#)

[-rrp mlsrv17 Option \[page 81\]](#)

[-zs mlsrv17 Option \[page 108\]](#)

[-rrp mlsrv17 Option \[page 81\]](#)

## 1.3.25 -nc mlsrv17 Option

Sets the maximum number of concurrent network connections.

≡ Syntax

```
mlsrv17 -c "connection-string" -nc connections ...
```

## Remarks

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

The default is 1024.

To limit the number of concurrent synchronizations for non-persistent HTTP/HTTPS, set `-nc` significantly higher than `-sm`. When the `-sm` limit is reached, the MobiLink server provides an HTTP error 503 (Service Unavailable) to the remote client. If the `-nc` limit is reached, however, a socket error is issued. The greater the difference between `-nc` and `-sm`, the more likely it is that the rejected connections will generate the HTTP 503 error instead of the less descriptive socket error. For example, set `-sm` to 100 and set `-nc` to 1000.

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.

## Related Information

[-sm mlsrv17 Option \[page 85\]](#)

[-sm mlsrv17 Option \[page 85\]](#)

## 1.3.26 -ncs mlsrv17 Option

Reads the first NCS (Native Component Supportability) configuration file in the path or current directory. The configuration file contains the settings required to connect to an SAP Diagnostic Agent.

### ≡ Syntax

```
mlsrv17 -c "connection-string" -ncs ...
```

## Applies to

Microsoft Windows and x64 Linux.

## Remarks

If the `ncs.conf` file is not present, then the Mobilink server uses NCS defaults

## Related Information

[MobiLink Server Logging and SAP Passports \[page 26\]](#)

[-ncsd mlsrv17 Option \[page 68\]](#)

[-ncsp mlsrv17 Option \[page 69\]](#)

### 1.3.27 -ncsd mlsrv17 Option

Reads the NCS (Native Component Supportability) configuration file in the specified directory. The configuration file contains the settings required to connect to an SAP Diagnostic Agent.

#### ≡ Syntax

```
mlsrv17 -c "connection-string" -ncsd directory ...
```

## Applies to

Windows and x64 Linux.

## Remarks

`directory` specifies the directory containing the NCS configuration file. If the `ncs.conf` file is not present in the specified directory, The MobiLink server uses NCS defaults.

## Related Information

[MobiLink Server Logging and SAP Passports \[page 26\]](#)

[-ncs mlsrv17 Option \[page 67\]](#)

[-ncsp mlsrv17 Option \[page 69\]](#)

## 1.3.28 -ncsp mlsrv17 Option

Specifies **name=value** pairs to configure the NCS (Native Component Supportability) library.

### ☰ Syntax

```
mlsrv17 -c "connection-string" -ncsd "name=value[;...]" ...
```

## Applies to

Microsoft Windows and x64 Linux.

## Remarks

Each name must be a valid configuration setting. See the SAP Diagnostic Agent documentation for valid names.

## Related Information

[MobiLink Server Logging and SAP Passports \[page 26\]](#)

[-ncs mlsrv17 Option \[page 67\]](#)

[-ncsd mlsrv17 Option \[page 68\]](#)

## 1.3.29 -notifier mlsrv17 Option

Starts the Notifier for server-initiated synchronization.

### ☰ Syntax

```
mlsrv17 -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.

## Related Information

[MobiLink Server Settings for Server-initiated Synchronization](#)  
[Server-side Settings Configured Using the Notifier Configuration File](#)  
[Notifiers](#)  
[Notifiers in a MobiLink Server Farm](#)  
[Notifier Properties](#)  
[MobiLink Server Settings for Server-initiated Synchronization](#)  
[Server-side Settings Configured Using the Notifier Configuration File](#)  
[Notifiers](#)  
[Notifiers in a MobiLink Server Farm](#)  
[Notifier Properties](#)

### 1.3.30 -o mlsrv17 Option

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

#### ☰, Syntax

```
mlsrv17 -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.

## Related Information

[-os mlsrv17 Option \[page 73\]](#)

[-dl mlsrv17 Option \[page 57\]](#)

[-ot mlsrv17 Option \[page 74\]](#)

[-on mlsrv17 Option \[page 71\]](#)

[-v mlsrv17 Option \[page 92\]](#)

[-os mlsrv17 Option \[page 73\]](#)

[-dl mlsrv17 Option \[page 57\]](#)

[-ot mlsrv17 Option \[page 74\]](#)

[-on mlsrv17 Option \[page 71\]](#)

[-v mlsrv17 Option \[page 92\]](#)

## 1.3.31 -on mlsrv17 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

```
mlsrv17 -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 message 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.

### i Note

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

This option cannot be used with the `-os` option.

## Related Information

[-o mlsrv17 Option \[page 70\]](#)

[-os mlsrv17 Option \[page 73\]](#)

[-ot mlsrv17 Option \[page 74\]](#)

[-v mlsrv17 Option \[page 92\]](#)

[-o mlsrv17 Option \[page 70\]](#)

[-ot mlsrv17 Option \[page 74\]](#)

[-on mlsrv17 Option \[page 71\]](#)

[-os mlsrv17 Option \[page 73\]](#)

[-v mlsrv17 Option \[page 92\]](#)

## 1.3.32 -oq mlsrv17 Option

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

### ≡ Syntax

```
mlsrv17 -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.



## 1.3.33 -os mlsrv17 Option

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

### ≡ Syntax

```
mlsrv17 -c "connection-string" -os size [ k | m ] ...
```

## Remarks

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

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

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

You cannot use this option with the `-on` option.

### i Note

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

## Related Information

[-o mlsrv17 Option \[page 70\]](#)

[-on mlsrv17 Option \[page 71\]](#)

[-ot mlsrv17 Option \[page 74\]](#)

[-v mlsrv17 Option \[page 92\]](#)

[-o mlsrv17 Option \[page 70\]](#)

[-on mlsrv17 Option \[page 71\]](#)

[-ot mlsrv17 Option \[page 74\]](#)

[-v mlsrv17 Option \[page 92\]](#)

## 1.3.34 -ot mlsrv17 Option

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

### ☰, Syntax

```
mlsrv17 -c "connection-string" -ot logfilename ...
```

### Remarks

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

### Related Information

[-on mlsrv17 Option \[page 71\]](#)

[-os mlsrv17 Option \[page 73\]](#)

[-v mlsrv17 Option \[page 92\]](#)

[-o mlsrv17 Option \[page 70\]](#)

[-on mlsrv17 Option \[page 71\]](#)

[-os mlsrv17 Option \[page 73\]](#)

[-v mlsrv17 Option \[page 92\]](#)

[-o mlsrv17 Option \[page 70\]](#)

## 1.3.35 -ppv mlsrv17 Option

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

### ☰, Syntax

```
mlsrv17 -c "connection-string" -ppv period ...
```

### Remarks

These values can provide insight into the state of the server, and are useful for determining the health and performance of the MobiLink server. For example, one could look at the DB\_CONNECTIONS and LONGEST\_DB\_WAIT values to look for potential problems with the -w option or in the synchronization scripts.

The values also provide an easy way to track system wide throughput measures, such as the number of rows uploaded or downloaded per second and the number of successful synchronizations per second.

The suggested period is 60 seconds.

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

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

The printed values can include the following information:

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

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

Printed value	Description
NUM_WAITING_CONS	The number of synchronizations currently waiting for the consolidated database.
NUM_WARNINGS	The total number of warnings.
PAGES_LOCKED_MAX	The number of pages in the memory cache.
PRIMARY_IS_KNOWN	Indicates if the primary server is known or not. Shows 0 if the server does not care what the primary server is. Shows 1 if the server knows what the primary server is. Shows 2 if the server does not know what the primary server is.
RAW_TCP_STAGE_LEN	The length of the network work queue.
SERVER_IS_PRIMARY	Indicates if the server is primary or secondary. Shows 1 if the server is primary, otherwise shows 0.
SIRT_NUM_LWP_HITS	The number of lightweight polls from remote task agents, indicating a notification.
SIRT_NUM_LWPS	The number of lightweight polls from remote task agents.
SIRT_NUM_REQUESTS	The number of remote task notifications currently outstanding.
STREAM_STAGE_LEN	The length of the high level network processing queue.
TCP_BYTES_READ	The total number of bytes ever read.
TCP_BYTES_WRITTEN	The total number of bytes ever written.
TCP_CONNECTIONS	The number of TCP connections currently opened.
TCP_CONNECTIONS_CLOSED	The total number of connections ever closed.
TCP_CONNECTIONS_OPENED	The total number of connections ever opened.
TCP_CONNECTIONS_REJECTED	The total number of connections ever rejected.
TRACKED_MEMORY	The amount of memory allocated by the server. Use this metric for non-Windows systems where the MEMORY_USED metric is unavailable. On Microsoft Windows systems, use the MEMORY_USED metric for increased accuracy.
VM_MEM_USE	The amount of memory used by any attached VMs.

## Example

Below is sample output showing the periodic monitoring values.

```
I. 2009-10-28 11:46:29. <Main> PERIODIC: TCP_CONNECTIONS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: PAGES_LOCKED_MAX: 13243
I. 2009-10-28 11:46:29. <Main> PERIODIC: TCP_CONNECTIONS_OPENED: 2
I. 2009-10-28 11:46:29. <Main> PERIODIC: TCP_CONNECTIONS_CLOSED: 2
I. 2009-10-28 11:46:29. <Main> PERIODIC: TCP_CONNECTIONS_REJECTED: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: TCP_BYTES_READ: 5137
I. 2009-10-28 11:46:29. <Main> PERIODIC: TCP_BYTES_WRITTEN: 4549
I. 2009-10-28 11:46:29. <Main> PERIODIC: ML_NUM_CONNECTED_CLIENTS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: CPU_USAGE: 3359375
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_COMMITS: 7
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_ROLLBACKS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_SUCCESS_SYNCNS: 1
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_FAILED_SYNCNS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_ERRORS: 2
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_WARNINGS: 3
I. 2009-10-28 11:46:29. <Main> PERIODIC: DB_CONNECTIONS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: RAW_TCP_STAGE_LEN: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: STREAM_STAGE_LEN: 5
I. 2009-10-28 11:46:29. <Main> PERIODIC: HEARTBEAT_STAGE_LEN: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: CMD_PROCESSOR_STAGE_LEN: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_ROWS_DOWNLOADED: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_ROWS_UPLOADED: 7
I. 2009-10-28 11:46:29. <Main> PERIODIC: FREE_DISK_SPACE: 124154904576
I. 2009-10-28 11:46:29. <Main> PERIODIC: LONGEST_DB_WAIT: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: LONGEST_SYNC: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: MEMORY_USED: 140275712
I. 2009-10-28 11:46:29. <Main> PERIODIC: SERVER_IS_PRIMARY: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_CONNECTED_SYNCNS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_CONNECTED_PINGS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_CONNECTED_FILE_XFERS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_CONNECTED_MONITORS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_CONNECTED_LISTENERS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_WAITING_CONNS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_SYNC_REQUEST: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_RECVING_UPLOAD: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_CONNECT: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_AUTH_USER: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_BEGIN_SYNC: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_APPLY_UPLOAD: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_PREP_FOR_DNLD: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_FETCH_DNLD: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_END_SYNC: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_SEND_DNLD: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_WAIT_FOR_DNLD_ACK: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_GET_DB_WORKER_FOR_ACK: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_CONNECT_FOR_ACK: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_IN_NON_BLOCKING_ACK: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: NUM_UPLOAD_CONNS_IN_USE: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: TRACKED_MEMORY: 56269577
I. 2009-10-28 11:46:29. <Main> PERIODIC: VM_MEM_USE: 517013504
I. 2009-10-28 11:46:29. <Main> PERIODIC: NOTIFIER_STAGE_LEN: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: SIRT_NUM_REQUESTS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: SIRT_NUM_LWPS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: SIRT_NUM_LWP_HITS: 0
I. 2009-10-28 11:46:29. <Main> PERIODIC: PRIMARY_IS_KNOWN: 0
```

### 1.3.36 -q mlsrv17 Option

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

#### ≡ Syntax

```
mlsrv17 -c "connection-string" -q ...
```

#### Remarks

Minimize the MobiLink server messages window.

### 1.3.37 -r mlsrv17 Option

Sets the maximum number of deadlock retries.

#### ≡ Syntax

```
mlsrv17 -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.

#### i Note

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

### 1.3.38 -rd mlsrv17 Option

Sets the maximum delay time between deadlock retries.

#### ≡ Syntax

```
mlsrv17 -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.

### i Note

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

## 1.3.39 -rp mlsrv17 Option

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

### ≡ Syntax

```
mlsrv17 -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.

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.

## Related Information

[-rrp mlsrv17 Option \[page 81\]](#)

[MobiLink Replay Utility \(mlreplay\) \[page 651\]](#)

[-rrp mlsrv17 Option \[page 81\]](#)

[MobiLink Replay Utility \(mlreplay\) \[page 651\]](#)



## 1.3.40 -rrp mlsrv17 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

```
mlsrv17 -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.

## Related Information

[-rp mlsrv17 Option \[page 80\]](#)

[-lsc mlsrv17 Option \[page 65\]](#)

[MobiLink Replay Utility \(mlreplay\) \[page 651\]](#)

[-rp mlsrv17 Option \[page 80\]](#)

[-lsc mlsrv17 Option \[page 65\]](#)

[MobiLink Replay Utility \(mlreplay\) \[page 651\]](#)

## 1.3.41 -s mlsrv17 Option

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

### ≡ Syntax

```
mlsrv17 -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 MobiLink server message log file as `rowset size`.

The default is 10.

## 1.3.42 -sl dnet mlsrv17 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

```
mlsrv17 -c "connection-string" -sl dnet( options ) ...
```

## Remarks

Sets options to pass directly to the .NET CLR. Some common options are:

Option	Description
<code>-D name=value</code>	Set an environment variable. For example: <pre>-Dsynchtype=far -Dextra_rows=yes</pre> For more information, see the .NET framework class <code>System.Environment</code> .
<code>-MLAutoLoadPath= path</code>	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.

Option	Description
<code>-MLDomConfigFile= file</code>	<p>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 <code>MLAutoLoadPath</code> for some other reason. To tell MobiLink where assemblies are located, use <code>-MLDomConfigFile</code> or <code>-MLAutoLoadPath</code>, but not both.</p> <p>When the file path referenced in the <code>-MLDomConfigFile</code> option refers to a file in a folder with a space in the name, such as "C:\Program Files\MyCompany\SyncServer\MlDomConfig.xml", place double quotes around the whole option: "<code>-MLDomConfigFile=C:\Program Files\MyCompany\SyncServer\MlDomConfig.xml</code>".</p>
<code>-MLStartClasses= classnames, ...</code>	At server startup, load and instantiate user-defined start classes in the order listed.
<code>-clrConGC</code>	Enable concurrent garbage collection in the CLR.
<code>-clrFlavor=( wks   svr )</code>	Flavor of the .NET CLR to load. The flavor is <code>svr</code> for server and <code>wks</code> for workstation. By default, <code>svr</code> is loaded.
<code>-clrVersion= version</code>	Version of the .NET CLR to load. This must be prefixed with <code>v</code> . For example, specifying <code>v4.0.30319</code> causes the MobiLink server to attempt to load the .NET CLR located in the <code>%SystemRoot%\Microsoft.NET\Framework\v4.0.30319</code> directory.

To use v4.0 or later assemblies, you need to explicitly add the `-clrVersion` option to make the MobiLink server load a v4.0 or later runtime. For example, `-clrVersion=v4.0.30319`.

Options must be enclosed in round brackets (parentheses) or curly brackets {braces}.

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

```
m1srv17 -sl dnet (?)
```

## Related Information

[Synchronization Scripts in Microsoft .NET \[page 541\]](#)

[Synchronization Scripts in Microsoft .NET \[page 541\]](#)

## 1.3.43 -sl java mlsrv17 Option

Sets the Java VM options and forces the Java VM to load on startup. This option is recommended when using Java scripting logic.

### ≡ Syntax

```
mlsrv17 -c "connection-string" -sl java( options ) ...
```

## Remarks

Sets options to indicate which Java VM to use and options to pass directly to the Java VM. Options include, but are not limited to, the following:

Option	Description
<code>-client</code> (Microsoft Windows only)	Use the client Java VM.
<code>-server</code> (Microsoft Windows only)	Use the server Java VM. This is the default.
<code>-jrepath path</code> (Microsoft Windows and macOS only)	Override the default Java Runtime Environment loaded by the MobiLink Server.
<code>-cp location;...</code>	Specify a set of directories or JAR files in which to search for classes. You can also use <code>-classpath</code> .
<code>-D name=value</code>	Set a system property. For example: <pre>-Dsynchtype=far -Dextra_rows=yes</pre>
<code>-DMLStartClasses= classname, ...</code>	At server startup, load and instantiate user-defined start classes in the order listed.
<code>-verbose [ :class   :gc   :jni ]</code>	Enable verbose output.
<code>-X vm-option</code>	Set a VM-specific option as described in the file <code>%SQLANY17%\Bin32\jre180\bin\server\Xusage.txt</code> or <code>%SQLANY17%\Bin64\jre180\bin\server\Xusage.txt</code> , for 32-bit and 64-bit platforms, respectively.

Options must be enclosed in round brackets (parentheses) or curly brackets {braces}.

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

```
mlsrv17 -sl java (?)
```

To display a list of Java options you can use, run the following command:

```
java
```

On UNIX and Linux, the `-cp` file paths must be separated by colons.

The `-jrepath` option can be used to override the default Java Runtime Environment loaded by the MobiLink Server on Windows and macOS. On Windows, the default JRE is located at `%SQLANY17%\Bin64\jre180`, and on macOS, the default JRE is located at `/System/Library/Frameworks/JavaVM.framework/Versions/A/JavaVM`. The syntax of the `-jrepath` is slightly different on Windows versus macOS.

On Windows, a directory is specified for the path, and the MobiLink Server will load `%path%\bin\server\jvm.dll` or `%path%\bin\client\jvm.dll`, based on whether `-server` (the default) or `-client` was specified within the `-sl java mlsrv17` option.

On macOS, the full path to the `libjli.dylib` shared library (including the shared library name) is specified for the path. On Linux and other UNIX platforms, 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 `libjvm.so` (`libjvm.sl` on HP-UX) shared library. The directory must be listed before any of the SQL Anywhere installation directories in the library path.

## Example

For example, on Microsoft Windows the following partial `mlsrv17` command line sets the Java VM classpath and enables Java VM system assertions.

```
mlsrv17 -sl java (-cp ;\myclasses; -esa) ...
```

On Microsoft Windows, the following partial `mlsrv17` command line sets the Java VM classpath and the Java VM `LDAP_SERVER` system property.

```
mlsrv17 -sl java ( -cp ;\myclasses; -DLdap_SERVER=mycorp-ldap ) ...
```

The following partial `mlsrv17` command line works on UNIX and Linux.

```
mlsrv17 -sl java { -cp .:$CLASSPATH:/opt/myclasses:/opt/my.jar: }
```

## Related Information

[Synchronization Script Writing in Java \[page 526\]](#)

[Synchronization Script Writing in Java \[page 526\]](#)

### 1.3.44 -sm mlsrv17 Option

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

#### ≡ Syntax

```
mlsrv17 -c "connection-string" -sm number ...
```

## Remarks

The default value is 0, which means the number of synchronizations is unlimited.

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 `mlsrv17 -w` option.
- The number of synchronizations doing task 2 is less than or equal to the setting for the `mlsrv17 -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.

## Related Information

[-w mlsrv17 Option \[page 96\]](#)

[-wu mlsrv17 Option \[page 98\]](#)

[-nc mlsrv17 Option \[page 66\]](#)

[MobiLink Server Java API Reference](#)

[-w mlsrv17 Option \[page 96\]](#)

[-wu mlsrv17 Option \[page 98\]](#)

[-nc mlsrv17 Option \[page 66\]](#)

### 1.3.45 -tc mlsrv17 Option

Sets a timeout threshold for long running SQL scripts.

≡ Syntax

```
mlsrv17 -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.

## Related Information

[-tf mlsrv17 Option \[page 87\]](#)

[-tf mlsrv17 Option \[page 87\]](#)

### 1.3.46 -tf mlsrv17 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

```
mlsrv17 -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.

## 1.3.47 -ts mlsrv17 Option

Sets up a MobiLink server tracing session.

### Syntax

```
mlsrv17 -c "connection-string" -ts session-name (session-option=option-value[;...])
```

The session name must be *logging*.

Session option	Option value
<i>events</i>	Comma separated list of system trace events. The supported events are Info, Warning, and Error.
<i>targets</i>	<code>target-type ( target-option = value [;...])</code> where <code>target-type</code> can only be <i>file</i> .

The target options are specified as name-value pairs. The target file can have the following options:

Target option name	Expected value	Description
<i>filename_prefix</i>	String	An ETD file name prefix with or without a path. All ETD files have the extension <code>.etd</code> . This parameter is required.
<i>max_size</i>	Integer	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.
<i>num_files</i>	Integer	The number of files where event tracing information is written, and it is used only if <code>max_size</code> is set. If all the files reach the maximum specified size, the MobiLink server starts overwriting the oldest file.



Target option name	Expected value	Description
<i>flush_on_write</i>	yes, true, no, false	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.
<i>compressed</i>	yes, true, no, false	A value that controls compression of the ETD file to conserve disk space. The default is false.

## Remarks

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

## Example

Following is an example of the `-ts` option:

```
-ts
logging(events=Info,warning,Error;targets=file(filename_prefix=mls_etd;max_size=1
000000;num_files=10;flush_on_write=true))
```

## Related Information

[Event Trace Data \(ETD\) File Management Utility \(dbmanageetd\)](#)

[Event Trace Data \(ETD\) File Management Utility \(dbmanageetd\)](#)

## 1.3.48 -tx mlsrv17 Option

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

### ≡ Syntax

```
mlsrv17 -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.

## Related Information

[-tu dbmsync Option](#)

[-tu dbmsync Option](#)

## 1.3.49 -ud mlsrv17 Option

Instructs MobiLink to run as a daemon.

≡ Syntax

```
mlsrv17 -c "connection-string" -ud ...
```

## Remarks

This option applies to UNIX and Linux platforms only.

## Related Information

[MobiLink Server Use Outside the Current Session \[page 29\]](#)

[MobiLink Server Use Outside the Current Session \[page 29\]](#)

## 1.3.50 -ui mlsrv17 Option

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

### ☰ Syntax

```
mlsrv17 -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.

## 1.3.51 -ux mlsrv17 Option

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

### ☰ Syntax

```
mlsrv17 -c "connection-string" -ux ...
```

### Remarks

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

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

On Windows, the MobiLink server messages window appears automatically.

### Related Information

[-q mlsrv17 Option \[page 79\]](#)

[-q mlsrv17 Option \[page 79\]](#)

## 1.3.52 -v mlsrv17 Option

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

### ≡ Syntax

```
mlsrv17 -c "connection-string" -v[ levels ] ...
```

## Remarks

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

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

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

The MobiLink server can be set to use different log verbosity for a targeted MobiLink user or remote ID. The MobiLink server checks the ml\_property table every five minutes and looks for verbose settings for a MobiLink user or remote ID.

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

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

**+**

Turn on all of the lowercase verbosity levels.

**c**

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

**e**

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

**F**

Show first-read errors. This setting logs errors when load-balancing devices check for server liveness by making connections that don't send any data. Use this option to verify that the load balancer is properly performing liveness checks.

See also the TCP/IP *ignore* option, specified with the -x mlsrv17 option.

**h**

Show the remote schema being synchronized.

**i**

Display the column values of each row uploaded. Use this option instead of -vr, which displays the column values of each row uploaded and downloaded, to reduce the amount of data being logged. Specifying -vi with -vq is the same as specifying -vr.

## m

Prints the duration of each synchronization and the duration of each synchronization phase to the log whenever a synchronization completes. The synchronization phases are shown below. They are the same as those displayed in the MobiLink 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: start_time: 2008-06-05 14:48:36.048
I. 2008-06-05 14:48:36. <1> PHASE: duration: 175
I. 2008-06-05 14:48:36. <1> PHASE: sync_request: 0
I. 2008-06-05 14:48:36. <1> PHASE: receive_upload: 19
I. 2008-06-05 14:48:36. <1> PHASE: get_db_worker: 0
I. 2008-06-05 14:48:36. <1> PHASE: connect: 18
I. 2008-06-05 14:48:36. <1> PHASE: authenticate_user: 51
I. 2008-06-05 14:48:36. <1> PHASE: begin_sync: 69
I. 2008-06-05 14:48:36. <1> PHASE: apply_upload: 0
I. 2008-06-05 14:48:36. <1> PHASE: prepare_for_download: 1
I. 2008-06-05 14:48:36. <1> PHASE: fetch_download: 4
I. 2008-06-05 14:48:36. <1> PHASE: wait_for_download_ack: 0
I. 2008-06-05 14:48:36. <1> PHASE: end_sync: 0
I. 2008-06-05 14:48:36. <1> PHASE: send_download: 10
I. 2008-06-05 14:48:36. <1> PHASE: get_db_worker_for_download_ack: 0
I. 2008-06-05 14:48:36. <1> PHASE: connect_for_download_ack: 0
I. 2008-06-05 14:48:36. <1> PHASE: nonblocking_download_ack: 0
```

### n

Show row-count totals per synchronization.

### o

Show SQL passthrough activity. (Deprecated)

### p

Show both remote and consolidated progress offsets per synchronization.

### q

Display the column values of each row downloaded. Use this option instead of `-vr`, which displays the column values of each row uploaded and downloaded, to reduce the amount of data being logged. Specifying `-vi` with `-vq` is the same as specifying `-vr`.

### r

Display the column values of each row uploaded or downloaded. To log only the column values of each row uploaded, use -vi. To log only the column values of each row downloaded, use -vq.

## R

For synchronizations only, show the remote ID in each log message. The MobiLink server adds the prefix yyyy-mm-dd hh:mm:ss. <sync\_id> (remote\_id,) to the log entries.

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

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

## S

Show the name of each synchronization script as it is invoked.

## t

Show the translated SQL that results from scripts that are written in ODBC canonical format. This level implies c. The following example shows the automatic translation of a statement for SQL Anywhere.

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

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

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

## u

Show undefined table scripts. This may help new users understand the synchronization process and the flow of events.

## U

For synchronizations only, shows the user name in each log message. The MobiLink server adds the prefix yyyy-mm-dd hh:mm:ss. <sync\_id> (,user\_name) to the log entries.

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

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

## Related Information

[MobiLink Synchronization Statistical Properties \[page 267\]](#)

[ml\\_add\\_property System Procedure \[page 605\]](#)

[-x mlsrv17 Option \[page 99\]](#)

[MobiLink Synchronization Statistical Properties \[page 267\]](#)

[ml\\_add\\_property System Procedure \[page 605\]](#)

[-x mlsrv17 Option \[page 99\]](#)

## 1.3.53 -w mlsrv17 Option

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

### ≡ Syntax

```
mlsrv17 -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`.

## Related Information

[-wm mlsrv17 Option \[page 97\]](#)

[-wu mlsrv17 Option \[page 98\]](#)

[-sm mlsrv17 Option \[page 85\]](#)

[-cn mlsrv17 Option \[page 55\]](#)

[-wm mlsrv17 Option \[page 97\]](#)

[-wu mlsrv17 Option \[page 98\]](#)



[-sm mlsrv17 Option \[page 85\]](#)

[-cn mlsrv17 Option \[page 55\]](#)

## 1.3.54 -wm mlsrv17 Option

Sets the maximum number of concurrent database worker threads.

### ≡ Syntax

```
mlsrv17 -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.

### Related Information

[Automatic Adjustment of Database Worker Threads \[page 187\]](#)

[-w mlsrv17 Option \[page 96\]](#)

[-wu mlsrv17 Option \[page 98\]](#)

[-sm mlsrv17 Option \[page 85\]](#)

[-cn mlsrv17 Option \[page 55\]](#)

[Automatic Adjustment of Database Worker Threads \[page 187\]](#)

[-w mlsrv17 Option \[page 96\]](#)

[-wu mlsrv17 Option \[page 98\]](#)

[-sm mlsrv17 Option \[page 85\]](#)

[-cn mlsrv17 Option \[page 55\]](#)

## 1.3.55 -wn mlsrv17 Option

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

### ☰ Syntax

```
mlsrv17 -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.

## 1.3.56 -wu mlsrv17 Option

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

### ☰ Syntax

```
mlsrv17 -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 mlsrv17 -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.

## Related Information

[-w mlsrv17 Option \[page 96\]](#)

[-wm mlsrv17 Option \[page 97\]](#)

[-sm mlsrv17 Option \[page 85\]](#)

[-w mlsrv17 Option \[page 96\]](#)

[-wm mlsrv17 Option \[page 97\]](#)

[-sm mlsrv17 Option \[page 85\]](#)

## 1.3.57 -x mlsrv17 Option

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

### ≡ Syntax

```
mlsrv17 -c "connection-string" -x protocol[protocol-options] [ -x  
protocol[protocol-options] ...] ...
```

```
protocol : tcpip | tls | http | https
```

```
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:

```
mlsrv17 -x tcpip(port=1234) -x http(port=2222)
```

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 TLS using RSA encryption.

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

### **TCP/IP options**

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

TCP/IP protocol option	Description
<code>collect_network_data={ yes   no }</code>	Enables synchronization scripts to read network information from each synchronization.
<code>host= hostname</code>	The host name or IP number on which the MobiLink server should listen. The default value is localhost.

TCP/IP protocol option	Description
<code>ignore= hostname</code>	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 <code>-x</code> 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.
<code>port= portnumber</code>	The socket port number on which the MobiLink server should listen. The default port is 2439, which is the IANA registered port number for the MobiLink server.

### Options for TCP/IP with transport layer security

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

TLS protocol options	Description
<code>collect_network_data={ yes   no }</code>	Enables synchronization scripts to read network information from each synchronization.
<code>e2ee_private_key= file</code>	<p>The PEM or DER encoded file containing the RSA private key. This option is required for end-to-end encryption to take effect.</p> <p>PEM and DER encoded files are created using the <code>create-key</code> utility.</p>
<code>e2ee_private_key_password= password</code>	<p>The password to the private key file. This option is required for end-to-end encryption to take effect.</p> <p>When this option is specified, the <code>e2ee_private_key</code> parameter must also be specified.</p> <p>To avoid making this password visible in the MobiLink server command line, use the <code>dbfhide</code> utility.</p>
<code>fips={yes no}</code>	If you are using the TLS protocol with RSA, you can specify <code>fips=yes</code> to accept connections using the TCP/IP protocol and FIPS-certified encryption algorithms. FIPS-certified connections use separate FIPS 140-2 certified software. Servers using RSA encryption without FIPS-certified encryption are compatible with clients using RSA that have the <code>fips</code> option enabled. Servers using RSA with the <code>fips</code> option enabled are compatible with clients using RSA that do not have the <code>fips</code> option enabled.
<code>host= hostname</code>	The host name or IP number on which the MobiLink server should listen. The default value is <code>localhost</code> .

TLS protocol options	Description
<code>identity= identity-file</code>	The path and file name of the identity file that is to be used for server authentication.
<code>identity_password= password</code>	<p>An optional parameter that specifies a password for the identity file.</p> <p>When this option is specified, the identity option must also be specified.</p> <p>To avoid making this password visible in the MobiLink server command line, use the dbfhide utility.</p>
<code>ignore= hostname</code>	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> .
<code>min_tls_version=ver</code>	Specify <code>min_tls_version</code> to use TLS version 1. The supported values for <code>ver</code> are <code>1.0</code> , <code>1.1</code> , and <code>1.2</code> (the default).
<code>port= portnumber</code>	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.
<code>trusted_certificates= certificate_file</code>	Use this option to ensure the client certificate is valid, then use the <code>NetworkData.ClientCertificates</code> API to further authenticate the certificate in the <code>authenticate_user</code> script. The <code>trusted_certificates</code> parameter takes the name of a file that contains a list of PEM-encoded X.509 trusted root certificates.

## HTTP options

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

HTTP options	Description
<code>buffer_size= number</code>	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.
<code>collect_network_data={ yes   no }</code>	Enables synchronization scripts to read network information from each synchronization.
<code>header_limit= number</code>	<p>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.</p>

HTTP options	Description
<code>host= hostname</code>	The host name or IP number on which the MobiLink server should listen. The default value is localhost.
<code>log_bad_request={ yes   no }</code>	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 <code>-vf</code> option. The default is no.
<code>port= portnumber</code>	The socket port number on which the MobiLink server should listen. The default port is 80.
<code>version= http-version</code>	The MobiLink server automatically detects the HTTP version used by a client. This parameter is a string specifying the default version of HTTP to use if the server cannot detect the version used by the client. You have a choice of 1.0 or 1.1. The default value is 1.1.

## HTTPS options

The HTTPS protocol uses RSA 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.

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

HTTPS options	Description
<code>buffer_size= number</code>	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.
<code>collect_network_data={ yes   no }</code>	Enables synchronization scripts to read network information from each synchronization.
<code>e2ee_private_key= file</code>	<p>The PEM or DER encoded file containing the RSA private key. This option is required for end-to-end encryption to take effect.</p> <p>PEM and DER encoded files are created using the <code>create-key</code> utility.</p>
<code>e2ee_private_key_password= password</code>	<p>The password to the private key file. This option is required for end-to-end encryption to take effect.</p> <p>When this option is specified, the <code>e2ee_private_key</code> option must also be specified.</p> <p>To avoid making this password visible in the MobiLink server command line, use the <code>dbfhide</code> utility.</p>

HTTPS options	Description
<code>fips={yes no}</code>	If you are using the TLS protocol with RSA, you can specify <code>fips=yes</code> to accept connections using the TCP/IP protocol and FIPS-certified encryption algorithms. FIPS-certified connections use separate FIPS 140-2 certified software. Servers using RSA encryption without FIPS-certified encryption are compatible with clients using RSA that have the <code>fips</code> option enabled. Servers using RSA with the <code>fips</code> option enabled are compatible with clients using RSA that do not have the <code>fips</code> option enabled.
<code>header_limit= number</code>	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.
<code>host= hostname</code>	The host name or IP number on which the MobiLink server should listen. The default value is localhost.
<code>identity= server-identity</code>	The path and file name of the identity file that is to be used for server authentication.
<code>identity_password= password</code>	An optional parameter that specifies a password for the identity file.  When this option is specified, the identity option must also be specified.  To avoid making this password visible in the MobiLink server command line, use the <code>dbfhide</code> utility.
<code>log_bad_request={ yes   no }</code>	When set to <code>yes</code> , the MobiLink server prints an error if it receives an incomplete or unexpected HTTP request. These errors are analogous to those printed by the <code>-vf</code> option. The default is <code>no</code> .
<code>min_tls_version=ver</code>	Specify <code>min_tls_version</code> to use TLS version 1. The supported values for <code>ver</code> are <code>1.0</code> , <code>1.1</code> , and <code>1.2</code> (the default).
<code>port= portnumber</code>	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.
<code>trusted_certificates= certificate_file</code>	Use this option to ensure the client certificate is valid, then use the <code>NetworkData.ClientCertificates</code> API to further authenticate the certificate in the <code>authenticate_user</code> script. The <code>trusted_certificates</code> parameter takes the name of a file that contains a list of PEM-encoded X.509 trusted root certificates.



HTTPS options	Description
<code>version= http-version</code>	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.

## Example

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

```
mksrv17 -c "DSN=SQL Anywhere 17 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:

```
mksrv17 -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:

```
mksrv17 -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:

```
mksrv17 -c "DSN=my_cons" -x https(identity=my_identity.id;
identity_password=my_id_pwd;e2ee_private_key=my_pk.pem;
e2ee_private_key_password=my_pk_pwd)
```

## trusted\_certificates example for Java

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

```
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.

```

mlsrv17 -c <connection_string> -x
https(collect_network_data=1;trusted_certificates=<certificate_file>) -sl java

```

```

mlsrv17 -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 {
    DBConnectionContext _cc;
    public OrderProcessor( DBConnectionContext 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.

```
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.

```
mlsrv17 -c <connection_string> -x  
https(collect_network_data=1;trusted_certificates=<certificate_file>) -sl dnet
```

```
mlsrv17 -c <connection_string> -x  
tls(collect_network_data=1;trusted_certificates=<certificate_file>) -sl dnet
```

## Related Information

[Transport Layer Security](#)

[Starting the MobiLink Server with Transport Layer Security \[page 197\]](#)

[Key Pair Generator Utility \(createkey\)](#)

[File Hiding Utility \(dbfhide\)](#)

[-v mlsrv17 Option \[page 92\]](#)

[Transport Layer Security](#)

[Starting the MobiLink Server with Transport Layer Security \[page 197\]](#)

[Key Pair Generator Utility \(createkey\)](#)

[File Hiding Utility \(dbfhide\)](#)

[-v mlsrv17 Option \[page 92\]](#)

## 1.3.58 -zf mlsrv17 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

```
mlsrv17 -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.

### 1.3.59 `-zp` mlsrv17 Option

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

#### ≡ Syntax

```
mlsrv17 -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.

### 1.3.60 `-zs` mlsrv17 Option

Specifies a MobiLink server name for `mlstop`.

#### ≡ Syntax

```
mlsrv17 -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.

## Related Information

[MobiLink Stop Utility \(mlstop\) \[page 648\]](#)

[MobiLink Stop Utility \(mlstop\) \[page 648\]](#)

### 1.3.61 -zt mlsrv17 Option

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

#### ≡ Syntax

```
mlsrv17 -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.

### 1.3.62 -zu mlsrv17 Option

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

#### ≡ Syntax

```
mlsrv17 -c "connection-string" -zu{ + | - } ...
```

#### Remarks

If this is supplied as `-zu+`, then unrecognized MobiLink user names are added automatically to the `ml_user` table if there is no user authentication script or if the user authentication scripts returned an `auth_value` of less

than 2999. If the argument is supplied as `-zu-`, or not supplied, the user is only added to the `ml_user` table if there is at least one user authentication script and the user is authenticated by the user authentication scripts.

The `-zu-` option cannot be specified with the `-zup` option.

The `-zu+` option is useful during development to register users automatically. The `-zu+` option should only be used in production if your authentication mechanism is the sole arbiter of which users can synchronize.

## Related Information

[Synchronizations from New Users](#)

[MobiLink Users in a Synchronization System](#)

[MobiLink User Authentication Utility \(mluser\) \[page 649\]](#)

[authenticate\\_user Connection Event \[page 354\]](#)

[Synchronizations from New Users](#)

[MobiLink Users in a Synchronization System](#)

[MobiLink User Authentication Utility \(mluser\) \[page 649\]](#)

[authenticate\\_user Connection Event \[page 354\]](#)

### 1.3.63 `-zup mlsrv17` Option

Specifies the default user authentication policy to be used to authenticate a user against the LDAP server.

#### ≡ Syntax

```
mlsrv17 -c "connection-string" -zup name
```

## Remarks

When a policy name is specified on the MobiLink server command line with this option, new MobiLink users that are not in the `ml_user` table are first authenticated against the LDAP server by using the specified default policy. Then optionally, the user authentication scripts are called if the `ldap_failover_to_std` property for the default policy is configured with a value of 1 or 2.

If the user is fully authenticated, then it is added into the `ml_user` table and the same user authentication policy is then used to subsequently authenticate this user.

This option cannot be used with `-zu-`. The MobiLink server issues an error if both `-zup` and `-zu-` are specified together.

#### i Note

The given default user authentication policy name must exist in the `ml_user_auth_policy` table; otherwise, the MobiLink server issues an error message and does not start.

## Related Information

[Synchronizations from New Users](#)

[MobiLink Users in a Synchronization System](#)

[ml\\_add\\_user\\_auth\\_policy System Procedure \[page 611\]](#)

[MobiLink User Authentication Utility \(mluser\) \[page 649\]](#)

[authenticate\\_user Connection Event \[page 354\]](#)

### 1.3.64 -zus mlsrv17 Option

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

#### ≡ Syntax

```
mlsrv17 -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.

### 1.3.65 -zw mlsrv17 Option

Controls which levels of warning message to display.

#### ≡ Syntax

```
mlsrv17 -c "connection-string" -zw levels
```

#### Remarks

MobiLink has five levels of warning messages:

Level	Description
0	Suppress all warning messages

Level	Description
1	Server and high ODBC level: warning messages when the MobiLink server starts
2	Synchronization and user level: warning messages when a synchronization starts
3	Schema level: warning messages when a MobiLink server is processing a client schema
4	Script and lower ODBC level: warning messages when a MobiLink server fetches, prepares, or executes scripts
5	Table or row level: warning messages when a MobiLink server performs table operations in an upload or download

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

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

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

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

## 1.3.66 -zwd mlsrv17 Option

Disables specific warning codes.

☰, Syntax

```
mlsrv17 -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.

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`.

### Related Information

[MobiLink Server Warning Messages](#)



## 1.3.67 -zwe mlsrv17 Option

Enables specific warning codes.

### ≡ Syntax

```
mlsrv17 -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.

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.

### Related Information

[MobiLink Server Warning Messages](#)

[MobiLink Server Warning Messages](#)

## 1.4 Synchronization Techniques

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.
- 

### In this section:

#### [Implementing Timestamp-based Downloads \[page 115\]](#)

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.

#### [Snapshot Synchronization \[page 119\]](#)

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.

#### [Partitioned Rows Among Remote Databases \[page 121\]](#)

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.

#### [Upload-only and Download-only Synchronizations \[page 125\]](#)

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.

#### [Unique Primary Keys \[page 126\]](#)

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.

#### [Conflict Handling Overview \[page 133\]](#)

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.

#### [Deletes \[page 142\]](#)

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.

#### [Failed Downloads \[page 144\]](#)

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.

#### [Download Acknowledgement \[page 147\]](#)

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.

#### [Result Sets from Stored Procedure Calls \[page 147\]](#)

You can download a result set from a stored procedure call.

#### [Self-referencing Tables \[page 149\]](#)

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.

#### [MobiLink Isolation Levels \[page 149\]](#)

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.

## 1.4.1 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.

### Context

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

### Procedure

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:

DBMS	last modified column
Adaptive Server Enterprise	<code>datetime</code>
IBM DB2 LUW	<code>timestamp NOT NULL GENERATED ALWAYS FOR EACH ROW ON UPDATE AS ROW CHANGE</code> <code>TIMESTAMP</code>
Microsoft SQL Server	<code>datetime</code>

#### i Note

Support for IBM DB2 consolidated databases is deprecated.

DBMS	last modified column
MySQL	timestamp default CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
Oracle	timestamp
SQL Anywhere	timestamp DEFAULT timestamp

- 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:

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

- `download_cursor` script:

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

### In this section:

#### [Last Download Times in Scripts \[page 117\]](#)

The last download timestamp is provided as a parameter to many MobiLink events.

#### [Daylight Savings Time Solutions \[page 119\]](#)

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

## Related Information

[Synchronization of Contacts in the Contact Sample](#)  
[Synchronization Logic Source Code](#)

### 1.4.1.1 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. Use named parameters instead.

#### 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.

## 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
```

### In this section:

[How Download Timestamps Are Generated and Used \[page 118\]](#)

MobiLink generates and uses a timestamp for timestamp-based downloads based on the following criteria.

## Related Information

[Script Parameters \[page 294\]](#)

### 1.4.1.1.1 How Download Timestamps Are Generated and Used

MobiLink generates and uses a timestamp for timestamp-based downloads based on the following criteria.

- 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 and can be viewed using the `SYSSYNC` system view.

For UltraLite remotes, the last download time is stored per publication in the `syspublication` system table.

## 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.

UltraLite also provides functionality to change the last download time from the remote with the following methods:

- `ULResetLastDownloadTime` method [UltraLite Embedded SQL]
- `ULConnection.ResetLastDownloadTime` method [UltraLite C++]
- `ULConnection.ResetLastDownloadTime` method [UltraLite.NET]

## Related Information

[SYSSYNC System View](#)

[syspublication System Table](#)

[modify\\_last\\_download\\_timestamp Connection Event \[page 457\]](#)

[generate\\_next\\_last\\_download\\_timestamp Connection Event \[page 435\]](#)

[modify\\_next\\_last\\_download\\_timestamp Connection Event \[page 460\]](#)

### 1.4.1.2 Daylight Savings Time Solutions

Daylight savings time can cause problems, even data loss, in a distributed database system if data is synchronized during the hour that the time changes. 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, choose from the following 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.

## 1.4.2 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.

## 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.
- 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.

## 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;
```

## Snapshot Example

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

1. There is no upload script. This reflects a business decision that products cannot be added at remote databases.
2. There is no download\_delete\_cursor, reflecting an assumption that products are not removed from the list.
3. The download\_cursor script selects the product identifier, price, and name of every current product. If the product is pre-existing, the price in the remote table is updated. If the product is new, a row is inserted in the remote table.

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

For another example of snapshot synchronization in a table with very few rows, see the MobiLink Contact sample.

## Related Information

[Partitioned Rows Among Remote Databases \[page 121\]](#)

[download\\_delete\\_cursor Scripts \[page 327\]](#)

[Scripts to Download Rows \[page 324\]](#)

[Synchronization of Sales Representatives in the Contact Sample](#)

### 1.4.3 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.

#### In this section:

##### [Disjoint Partitioning with MobiLink \[page 122\]](#)

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.

##### [Partitions with Overlaps \[page 123\]](#)

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.

##### [Partitioned Foreign Key Tables \[page 124\]](#)

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.

## 1.4.3.1 Disjoint Partitioning with MobiLink

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.

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

### 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.

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.

The download\_cursor script for this table is as follows.

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

## Related Information

[Primary Key Pools \[page 131\]](#)

[Synchronization of Customers in the Contact Sample](#)

[Synchronization of Contacts in the Contact Sample](#)

[Implementing Timestamp-based Downloads \[page 115\]](#)

[download\\_cursor Table Event \[page 393\]](#)

[download\\_delete\\_cursor Table Event \[page 396\]](#)

### 1.4.3.2 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:

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

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

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

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

The download\_delete\_cursor script is as follows.

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

This script deletes all approved rows from the remote database.

### 1.4.3.3 Partitioned Foreign Key Tables

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 in the MobiLink Contact sample.

## Related Information

[Synchronization of Contacts in the Contact Sample Partitions with Overlaps \[page 123\]](#)

### 1.4.4 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.

#### **i** Note

You can also specify upload-only or download-only if you create a synchronization model in SQL Central.

## SQL Anywhere Remote Databases

### Upload

To perform upload-only synchronization, use the `-uo dbmlsync` option or the `UploadOnly (uo)` extended option.

### Download

To perform download-only synchronization, use the `-ds dbmlsync` option or the `DownloadOnly (ds)` extended option.

SQL Anywhere remote databases can also use download-only publications. This approach to downloads is different from download-only synchronizations.

## UltraLite Remote Databases

### Upload

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

### Download

To perform download-only synchronization, use the `Download Only` synchronization parameter.

## Related Information

[Download-only Publications](#)  
[-uo dbmlsync Option](#)  
[UploadOnly \(uo\) Extended Option](#)

[-ds dbmsync Option](#)  
[DownloadOnly \(ds\) Extended Option](#)  
[Upload Only Synchronization Parameter](#)  
[Download Only Synchronization Parameter](#)

## 1.4.5 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.

Following are some solutions for generating unique primary keys across all synchronized databases:

- composite keys
- UUIDs
- GLOBAL AUTOINCREMENT
- primary key pools

### In this section:

#### [Composite Keys \[page 127\]](#)

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.

#### [UUIDs \[page 127\]](#)

On the clients, 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.

#### [GLOBAL AUTOINCREMENT \[page 128\]](#)

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.

#### [Primary Key Pools \[page 131\]](#)

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.

## 1.4.5.1 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.

### Related Information

[Remote IDs](#)

## 1.4.5.2 UUIDs

On the clients, 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.

### Example

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

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

### Related Information

[The NEWID Default](#)

[Primary Key Uniqueness in UltraLite](#)

[UNIQUEIDENTIFIER Data Type](#)

[NEWID Function \[Miscellaneous\]](#)

## 1.4.5.3 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.

### In this section:

[Using GLOBAL AUTOINCREMENT Columns \[page 128\]](#)

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

[DEFAULT GLOBAL AUTOINCREMENT \[page 129\]](#)

You can set default values in your database by selecting the column properties in SQL Central, or by including the DEFAULT GLOBAL AUTOINCREMENT phrase in a CREATE TABLE or ALTER TABLE statement.

[Global Database IDs \[page 130\]](#)

When deploying an application, you must assign a different identification number to each database.

### 1.4.5.3.1 Using GLOBAL AUTOINCREMENT Columns

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

#### Context

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.

#### Procedure

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.

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.

## Results

The column is set as a GLOBAL AUTOINCREMENT column.

## Related Information

[DEFAULT GLOBAL AUTOINCREMENT \[page 129\]](#)

[Global Database IDs \[page 130\]](#)

### 1.4.5.3.2 DEFAULT GLOBAL AUTOINCREMENT

You can set default values in your database by selecting the column properties in SQL Central, or by including the DEFAULT GLOBAL AUTOINCREMENT phrase in a CREATE TABLE or ALTER TABLE statement.

Optionally, the partition size can be specified in parentheses immediately following the AUTOINCREMENT keyword. The partition size may be any positive integer, although the partition size is generally chosen so that the supply of numbers within any one partition is rarely, if ever, exhausted.

For columns of type INT or UNSIGNED INT, the default partition size is  $2^{16} = 65536$ ; for columns of other types the default partition size is  $2^{32} = 4294967296$ . Since these defaults may be inappropriate, especially if your column is not of type INT or BIGINT, it is best to specify the partition size explicitly.

For example, the following SQL statement creates a simple table with two columns: an integer that holds a customer identification number and a character string that holds the customer's name. The partition size is set to 5000, which would be appropriate for an application database where few new rows are inserted in each remote database.

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

## Related Information

[CREATE TABLE Statement](#)

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

### 1.4.5.3.3 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.

### 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  $p_n + 1$  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  $p_n + 1$ , 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  $p_n + 1$  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.

You can set an event handler to automatically notify the database administrator (or perform some other action) when the partition is nearly exhausted.

## Example

In a SQL Anywhere database, the following statement sets the database identification number to 20.

```
SET OPTION PUBLIC.global_database_id = 20
```

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

## Related Information

[Primary Key Pools \[page 131\]](#)

[Trigger Conditions for Events](#)

[global\\_database\\_id Option](#)

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

### 1.4.5.4 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.

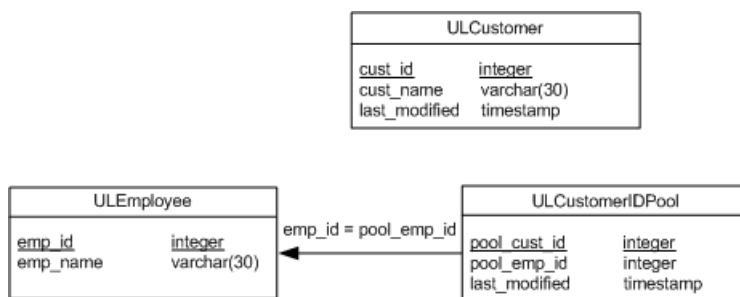
1. Add a new table to the consolidated database and to each remote database to hold the new primary key pool. Apart from a column for the unique value in the consolidated database, these tables should contain a column for a user name, to identify who has been given the right to assign the value.
2. In 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.
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;
}
```

## Related Information

[Implementing Timestamp-based Downloads \[page 115\]](#)

### 1.4.6 Conflict Handling Overview

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.

#### ⚠ 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.

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 other upload events that are described.

During the download stage of a synchronization, no conflicts arise in the remote database. If a downloaded row contains a new primary key, the values are inserted into a new row. If the primary key matches that of a pre-existing row, the values in the row are updated.

## Example

User1 starts with an inventory of ten items, and then sells three and updates the Remote1 inventory value to seven items. User2 sells four items and updates the Remote2 inventory to six. When Remote1 synchronizes, the consolidated database is updated to seven. When Remote2 synchronizes, a conflict is detected because the value of the inventory is no longer ten. To resolve this conflict programmatically, you need three row values:

1. The current value in the consolidated database.
2. The new row value that Remote2 uploaded.
3. The old row value that Remote2 obtained during the last synchronization.

In this case, the business logic could use the following to calculate the new inventory value and resolve the conflict:

```
current consolidated - (old remote - new remote)
-> 7 - (10-6) = 3
```

### In this section:

#### [Conflict Detection \[page 134\]](#)

When a MobiLink client sends an updated row to the MobiLink server, it includes the new updated values (the post-image), and 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.

#### [Conflict Resolution \[page 137\]](#)

You have several options for resolving conflicts.

## Related Information

[Synchronization of Products in the Contact Sample](#)

### 1.4.6.1 Conflict Detection

When a MobiLink client sends an updated row to the MobiLink server, it includes the new updated values (the post-image), and 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.

### In this section:

#### [Conflict Detection with upload\\_fetch or upload\\_fetch\\_column\\_conflict Scripts \[page 135\]](#)

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.

#### [Conflict Detection with upload\\_update Scripts \[page 137\]](#)

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.

### 1.4.6.1.1 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.

#### **i** 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):

```
SELECT pk1, pk2, ...col1, col2, ...
FROM table-name
WHERE pk1 = {ml.r.pk1} AND pk2 = {ml.r.pk2} ...
AND col1 = {ml.o.col1} AND col2 = {ml.o.col2} ...
```

#### **i** 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 = {mlr.pk1} AND pk2 = {mlr.pk2} ...
```

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.

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`.

## Related Information

[Conflict Resolution with `resolve\_conflict` Scripts \[page 138\]](#)

[`upload\_fetch` Table Event \[page 502\]](#)

[`upload\_fetch\_column\_conflict` Table Event \[page 504\]](#)

### 1.4.6.1.2 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.

## Related Information

[Conflict Resolution with `upload\_update` Scripts \[page 139\]](#)

### 1.4.6.2 Conflict Resolution

You have several options for resolving conflicts.

- Resolve conflicts as they occur using temporary or permanent tables and a `resolve_conflict` script.
- Resolve conflicts as they occur using an `upload_update` script.
- Resolve all conflicts at once using a table's `end_upload` script.

#### In this section:

[Conflict Resolution with `resolve\_conflict` Scripts \[page 138\]](#)

When the MobiLink server detects a conflict using an `upload_fetch` script, a number of events take place.

[Conflict Resolution with `upload\_update` Scripts \[page 139\]](#)

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.

## Related Information

[end\\_upload Table Event \[page 426\]](#)

### 1.4.6.2.1 Conflict Resolution with resolve\_conflict Scripts

When the MobiLink server detects a conflict using an upload\_fetch script, a number of 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.
- 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.
- 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.

#### Example

In the following example, you create scripts for six events and then you create a stored procedure.

- In the begin\_synchronization script, you create two temporary tables called contact\_new and contact\_old. (You could also do this in the begin\_connection script.)
- The upload\_fetch script detects the conflict.
- When there is a conflict, the upload\_old\_row\_insert and upload\_new\_row\_insert scripts populate the two temporary tables with the new and old data uploaded from the remote database.
- The resolve\_conflict script calls the stored procedure MLResolveContactConflict to resolve the conflict.

Event	Script
begin_synchronization	<pre>CREATE TABLE #contact_new(   id INTEGER,   location CHAR(36),   contact_date DATE); CREATE TABLE #contact_old(   id INTEGER,   location CHAR(36),   contact_date DATE)</pre>
upload_fetch	<pre>SELECT id, location, contact_date FROM contact WHERE id = {ml r.id}</pre>
upload_old_row_insert	<pre>INSERT INTO #contact_new( id, location, contact_date ) VALUES ( {ml r.id}, {ml r.location}, {ml r.contact_date} )</pre>

Event	Script
upload_new_row_insert	<pre>INSERT INTO #contact_old( id, location, contact_date ) VALUES ( {ml r.id}, {ml r.location}, {ml r.contact_date} )</pre>
resolve_conflict	<pre>CALL MLResolveContactConflict( )</pre>
end_synchronization	<pre>DROP TABLE #contact_new; DROP TABLE #contact_old</pre>

The stored procedure MLResolveContactConflict is as follows:

```
CREATE PROCEDURE MLResolveContactConflict( )
BEGIN
--update the consolidated database only if the new contact date
--is later than the existing contact date
UPDATE contact c
SET c.contact_date = cn.contact_date
FROM #contact_new cn
WHERE c.id = cn.id
AND cn.contact_date > c.contact_date;
--cleanup
DELETE FROM #contact_new;
DELETE FROM #contact_old;
END
```

## Related Information

[upload\\_old\\_row\\_insert Table Event \[page 510\]](#)

[upload\\_new\\_row\\_insert Table Event \[page 508\]](#)

[resolve\\_conflict Table Event \[page 483\]](#)

### 1.4.6.2.2 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:

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

The UpdateProduct stored procedure could be:

```
CREATE PROCEDURE UpdateProduct (
    @id INTEGER,
    @preName VARCHAR(20),
    @preDesc VARCHAR(200),
    @postName VARCHAR(20),
    @postDesc VARCHAR(200) )
BEGIN
    UPDATE product
    SET name = @postName, description = @postDesc
    WHERE id = @id
        AND name = @preName
        AND description = @preDesc
    IF @@rowcount=0 THEN
        // A conflict occurred: handle resolution here.
    END IF
END
```

This approach is often easier to maintain than resolving conflicts with resolve\_conflict scripts because there is only one script to maintain and all the logic is contained in one stored procedure. However, the code of the stored procedure may be complicated if the tables columns are nullable or if they contain BLOBs or CLOBs. Also, some RDBMSs that are supported MobiLink consolidated databases have limitations on the size of values that can be passed to stored procedures.

## Example

The following stored procedure, sp\_update\_my\_customer, contains logic for conflict detection and resolution. It accepts old column values and new column values. This example uses SQL Anywhere features. The script could be implemented as follows.

```
{CALL sp_update_my_customer(
    {ml o.cust_1st_pk},
    {ml o.cust_2nd_pk},
    {ml o.first_name},
    {ml o.last_name},
    {ml o.nullable_col},
    {ml o.last_modified},
    {ml r.first_name},
    {ml r.last_name},
    {ml r.nullable_col},
    {ml r.last_modified}
)}
CREATE PROCEDURE sp_update_my_customer(
    @cust_1st_pk      INTEGER,
    @cust_2nd_pk      INTEGER,
    @old_first_name   VARCHAR(100),
    @old_last_name    VARCHAR(100),
    @old_nullable_col VARCHAR(20),
    @old_last_modified DATETIME,
    @new_first_name   VARCHAR(100),
    @new_last_name    VARCHAR(100),
    @new_nullable_col VARCHAR(20),
    @new_last_modified DATETIME
)
BEGIN
    DECLARE @current_last_modified DATETIME;
    // Detect a conflict by checking the number of rows that are
    // affected by the following update. The WHERE clause compares
    // old values uploaded from the remote 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 nullable_col IS NOT DISTINCT FROM @old_nullable_col
  AND last_modified     = @old_last_modified;
...
// Use the @@rowcount global variable to determine
// the number of rows affected by the update. If @@rowcount=0,
// a conflict has occurred. In this example, the database with
// the most recent update wins the conflict. If the consolidated
// database wins the conflict, it retains its current values
// and no action is taken.
IF( @@rowcount = 0 ) THEN
// A conflict has been detected. To resolve it, use business
// logic to determine which values to use, and update the
// consolidated database with the final values.
  SELECT last_modified INTO @current_last_modified
  FROM my_customer WITH( HOLDLOCK )
  WHERE cust_1st_pk=@cust_1st_pk
    AND cust_2nd_pk=@cust_2nd_pk;
  IF( @new_last_modified > @current_last_modified ) THEN
// The remote 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;

```

## Related Information

[Conflict Detection with upload\\_update Scripts \[page 137\]](#)

[Conflict Resolution with resolve\\_conflict Scripts \[page 138\]](#)

[SQL Variables](#)

[upload\\_update Table Event \[page 522\]](#)

[COALESCE Function \[Miscellaneous\]](#)

## 1.4.7 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.

Use one of the following strategies to record deleted rows:

### 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.

### In this section:

#### [Temporarily Stopping the Synchronization of Deletes \[page 143\]](#)

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.

## Related Information

[download\\_delete\\_cursor Scripts \[page 327\]](#)

## 1.4.7.1 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.

### Context

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.

### Procedure

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.

### Related Information

[STOP SYNCHRONIZATION DELETE Statement \[MobiLink\]](#)  
[STOP SYNCHRONIZATION DELETE Statement \[UltraLite\]](#)  
[START SYNCHRONIZATION DELETE Statement \[MobiLink\]](#)  
[START SYNCHRONIZATION DELETE Statement \[UltraLite\]](#)

## 1.4.8 Failed Downloads

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.

#### In this section:

[Resumption of Failed Downloads \[page 144\]](#)

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 can be used for a restartable download.

### Related Information

[SendDownloadAck \(sa\) Extended Option](#)

[Send Download Acknowledgement Synchronization Parameter](#)

#### 1.4.8.1 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 can be used for a restartable download.

The MobiLink 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. The `-ds mlsrv17` option can be used with restartable downloads to specify the maximum amount of data on disk that the MobiLink server can use to store all restartable downloads.

### **i Note**

For a resumable download attempt to succeed, the remote database must connect to the same MobiLink server. If the remote database connects to a different MobiLink server or if the download has been pushed out of the same MobiLink server, the attempted resume will fail.

## **When Should You Resume Failed Downloads?**

The need for resumable downloads increases as network quality deteriorates and download sizes increase. If you only do small synchronizations or if you synchronize on a LAN or WLAN, then you likely do not need to resume downloads.

## **SQL Anywhere remote databases**

When synchronization fails during a download, the downloaded data is not applied to the remote database, however, the successfully transmitted portions of the download are stored in a temporary file on the remote device. Dbmlsync uses this file to avoid lengthy retransmission of data, and to recover from download failure.

There are three ways to implement this functionality. For all options, dbmlsync aborts and the resumed download fails if there is any new data to be uploaded.

### **-dc**

After a download fails, use `-dc` the next time you start dbmlsync to resume the download. If part of the failed download was transmitted, the MobiLink server only transmits the remainder of the download.

### **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.

### **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.

## **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 might 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.

- To resume the download, set the Resume Partial Download synchronization parameter to true and synchronize again.  
The restarted synchronization does not perform an upload, and downloads only those changes that would have been downloaded by the failed download. That is, it completes the failed download but does not synchronize changes made since the previous attempt. To get those changes, you need to synchronize again once the failed download has completed, or call Rollback Partial Download and synchronize with Resume Partial Download set to false.  
When you restart the download, many of the synchronization parameters from the failed synchronization are used again automatically. For example, the publications parameter is ignored: the synchronization downloads those publications requested on the initial download. The only parameters that must be set are the Resume Partial Download parameter (which must be set to true) and the User Name parameter. In addition, settings for the following parameters are obeyed, if set:
  - Keep Partial Download
  - DisableConcurrency
  - Observer
  - User Data
- To roll back the changes from the failed download without resuming synchronization, call the appropriate method or function to roll back the changes. This function is ULRollbackPartialDownload function for Embedded SQL. For UltraLite components, it is a method on the Connection object.

#### **UltraLite.NET**

ULConnection.RollbackPartialDownload method [UltraLite.NET]

#### **Embedded SQL**

ULRollbackPartialDownload method [UltraLite Embedded SQL]

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.

#### **i Note**

If the send\_download\_ack synchronization parameter is set to true, the setting is ignored for the resumed download.

## **Related Information**

[-ds mlsrv17 Option \[page 58\]](#)

[-dc dbmlsync Option](#)

[ContinueDownload \(cd\) Extended Option](#)

[sp\\_hook\\_dbmlsync\\_set\\_extended\\_options](#)

[sp\\_hook\\_dbmlsync\\_end](#)

[Keep Partial Download Synchronization Parameter](#)

[Partial Download Retained Synchronization Parameter](#)

[Resume Partial Download Synchronization Parameter](#)

## 1.4.9 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.

Download acknowledgement 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.

### i Note

Download acknowledgement cannot be used with resumable downloads.

## Related Information

[Resumption of Failed Downloads \[page 144\]](#)

[publication\\_nonblocking\\_download\\_ack Connection Event \[page 472\]](#)

[nonblocking\\_download\\_ack Connection Event \[page 466\]](#)

[SendDownloadAck \(sa\) Extended Option](#)

[Send Download Acknowledgement Synchronization Parameter](#)

## 1.4.10 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 TIMESTAMP  
)
```

The `download_cursor` table script might look as follows:

```
SELECT pk, col1, col2
```

```
FROM MyTable
  WHERE last_modified >= {ml s.last_table_download}
  AND employee = {ml s.username}
```

If you want your downloads to MyTable to use more sophisticated business logic, you can now create your script as follows, where DownloadMyTable is a stored procedure taking two parameters (last-download timestamp and MobiLink user name) and returning a result set. (This example uses an ODBC calling convention for portability.):

```
{call DownloadMyTable( {ml s.last_table_download}, {ml s.username} )}
```

The following are some simple examples for each supported consolidated database. Consult the documentation for your consolidated database for full details.

The following example works with SQL Anywhere, Adaptive Server Enterprise, Microsoft Azure, and Microsoft SQL Server.

```
CREATE PROCEDURE DownloadMyTable
  @last_dl_ts DATETIME,
  @u_name VARCHAR( 128 )
AS
BEGIN
  SELECT pk, col1, col2
  FROM MyTable
  WHERE last_modified >= @last_dl_ts
  AND employee = @u_name
END
```

For Oracle, the result set can be returned by a REF CURSOR defined in a stored procedure. However, when using the SQL Anywhere 17 - Oracle ODBC driver, the REF CURSOR parameter should be defined as the last one in the parameter list of the stored procedure. The REF CURSOR parameter can be defined as OUT or IN OUT. The following stored procedure works with Oracle.

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

```
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.

### **i** Note

Support for IBM DB2 consolidated databases is deprecated.

```
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;
```

## 1.4.11 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.

Self-referencing 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 `dbmsync -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.

### Related Information

[-tu dbmsync Option](#)

## 1.4.12 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 for downloads. 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. When a download transaction performs no updates, which is recommended, snapshot isolation eliminates the problem of downloads being directly blocked by other transactions.

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 Consolidated Databases

SQL Anywhere version 10 and later supports 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 snapshot isolation for SQL Anywhere is recommended to avoid the improbable scenario of missing rows on download, especially if you expect to have uncommitted changes during synchronization that are later rolled back. For example, at SQL\_TXN\_READ\_COMMITTED isolation level, SQL Anywhere does not block queries on rows where an uncommitted change to the row would change it from being selected to not being selected. If such a change is uncommitted during a synchronization with a remote database, thereby causing the row to be missed by the download cursor, and the change is subsequently rolled back, the row may never be downloaded to that remote database.

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.

You can enable snapshot isolation for upload with the `mlsrv17 -esu` option, and disable snapshot isolation with the `mlsrv17 -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.

## 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.

By default, the MobiLink server assumes datarows locking, which is non-blocking, so it uses the oldest open transaction start time for the next last download timestamp to not miss rows on download. If datarows locking is not used, then you can use the `-dr mlsrv17` option.

## 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 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 Consolidated Databases

Microsoft SQL Server 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 Microsoft SQL Server consolidated database. If snapshot is not enabled, MobiLink uses the default `SQL_TXN_READ_COMMITTED`. See your Microsoft SQL Server documentation for details.

You can enable snapshot isolation for upload with the `mlsrv17 -esu` option, and disable snapshot isolation with the `mlsrv17 -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.

In Microsoft Azure consolidated databases, by default, uncommitted operations (inserts, updates, and deletes) do not prevent other connections from accessing the same tables. Because of this behavior, do not use the `-dsd` option (disable snapshot isolation for download) when you use timestamp-based download logic for your synchronization. Doing so can cause data inconsistencies.

To use snapshot isolation on Microsoft SQL Server, the user ID that you use to connect the MobiLink server to the database must have permission to access the Microsoft 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.

## Related Information

[How to Enable Snapshot Isolation](#)

[The Synchronization Process](#)

[Isolation Levels and Consistency](#)

[Snapshot Isolation](#)

[-dsd mlsrv17 Option \[page 59\]](#)

[-esu mlsrv17 Option \[page 62\]](#)

[-dr mlsrv17 Option \[page 58\]](#)

[-dt mlsrv17 Option \[page 60\]](#)

## 1.5 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. Your consolidated database can be one of the following ODBC-compliant RDBMSs:

- SAP SQL Anywhere
- SAP Adaptive Server Enterprise
- IBM DB2 LUW (deprecated)
- Microsoft SQL Server
- MySQL
- Oracle
- SAP HANA
- SAP IQ

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

## Synchronizing to Other Data Sources

Your MobiLink environment must have a database that has been set up as a consolidated database. However, you can use direct row handling to 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.

## 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.

### In this section:

#### [How Remote Tables Relate to Consolidated Tables \[page 154\]](#)

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.

#### [Consolidated Database Setup \[page 155\]](#)

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 `%SQLANY17%\MobiLink\setup`.

#### [RDBMS-Dependent Synchronization Scripts \[page 157\]](#)

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

#### [Synchronization of Spatial Data \[page 158\]](#)

The MobiLink server supports synchronization of tables containing columns with spatial data types.

#### [Adaptive Server Enterprise Consolidated Database \[page 163\]](#)

MobiLink supports Adaptive Server Enterprise consolidated databases.

#### [IBM DB2 LUW Consolidated Database \[page 165\]](#)

MobiLink supports IBM DB2 LUW for Linux and Windows.

#### [Microsoft SQL Server and Microsoft Azure Consolidated Databases \[page 168\]](#)

MobiLink supports Microsoft SQL Server and Microsoft Azure consolidated databases.

#### [MySQL Consolidated Database \[page 170\]](#)

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

#### [Oracle Consolidated Database \[page 173\]](#)

MobiLink supports Oracle consolidated databases.

#### [SAP HANA Consolidated Database \[page 179\]](#)

MobiLink supports SAP HANA consolidated databases.

[SQL Anywhere Consolidated Database \[page 181\]](#)

MobiLink supports SQL Anywhere consolidated databases.

[SAP IQ Consolidated Database \[page 182\]](#)

MobiLink supports SAP IQ consolidated databases.

## Related Information

[Supported Platforms](#)

[Recommended ODBC Drivers For MobiLink](#)

[SAP SQL Anywhere Supported Platforms and Engineering Support Status](#)

## 1.5.1 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 can 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.

## Related Information

[MobiLink Data Mappings Between Remote and Consolidated Databases \[page 662\]](#)

### 1.5.2 Consolidated Database Setup

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 `%SQLANY17%\MobiLink\setup`.

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.

#### **i** 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.

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

## 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.

#### In this section:

[MobiLink System Database \[page 156\]](#)

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.

[MobiLink Server System Tables \[page 156\]](#)

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.

## Related Information

[MobiLink System Setup](#)

[MobiLink Server System Procedures \[page 582\]](#)

[Required Privileges for MobiLink Server \[page 23\]](#)

[ODBC Drivers for MobiLink \[page 710\]](#)

[Adaptive Server Enterprise Consolidated Database \[page 163\]](#)

[IBM DB2 LUW Consolidated Database \[page 165\]](#)

[Microsoft SQL Server and Microsoft Azure Consolidated Databases \[page 168\]](#)

[MySQL Consolidated Database \[page 170\]](#)

[Oracle Consolidated Database \[page 173\]](#)

[SAP IQ Consolidated Database \[page 182\]](#)

[SQL Anywhere Consolidated Database \[page 181\]](#)

## 1.5.2.1 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.

### i 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.

## 1.5.2.2 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.

## 1.5.3 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.

For specific information about each type of consolidated database, see the section for your RDBMS.

### .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.

### 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 `mlsrv17 -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.

### 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 the documentation for MobiLink data mappings.

## Related Information

[Synchronization Scripts \[page 288\]](#)

[Java Synchronization Logic \[page 528\]](#)

[Synchronization Scripts in Microsoft .NET \[page 541\]](#)

[MobiLink Data Mappings Between Remote and Consolidated Databases \[page 662\]](#)

[Synchronization Events \[page 332\]](#)

[SQL Anywhere Consolidated Database \[page 181\]](#)

[Adaptive Server Enterprise Consolidated Database \[page 163\]](#)

[IBM DB2 LUW Consolidated Database \[page 165\]](#)

[Microsoft SQL Server and Microsoft Azure Consolidated Databases \[page 168\]](#)

[MySQL Consolidated Database \[page 170\]](#)

[Oracle Consolidated Database \[page 173\]](#)

[SAP IQ Consolidated Database \[page 182\]](#)

[-b mlsrv17 Option \[page 51\]](#)

## 1.5.4 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 Azure
- Microsoft SQL Server
- IBM DB2 LUW

### i Note

Support for IBM DB2 consolidated databases is deprecated.

- 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. Use named parameters instead. If question marks are used, each spatial column must contain two question marks. The first one is for the WKB value and the second is for the SRID value.

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

```
{ml r.column_name:data}
```

```
{ml r.column_name:srid}
```

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

## 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:

```
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
```

```

        )
    );

```

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

```

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:

```

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.

#### In this section:

##### [Upload and Download Scripts \[page 161\]](#)

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.

## Related Information

### [Spatial Data](#)

### [Named Script Parameters \[page 294\]](#)

### [User-defined Named Parameters \[page 310\]](#)



## 1.5.4.1 Upload and Download Scripts

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

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

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

### SQL Anywhere Sample Scripts

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

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

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

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

### Microsoft SQL Server Sample Scripts

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

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

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

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

## Oracle Sample Scripts

The following is a sample upload\_insert script for Oracle:

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

The following is a sample download\_cursor script for Oracle:

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

## IBM DB2 Sample Scripts (Deprecated)

### i Note

Support for IBM DB2 consolidated databases is deprecated.

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
```

## Related Information

[SQL Anywhere Consolidated Database \[page 181\]](#)

[Microsoft SQL Server and Microsoft Azure Consolidated Databases \[page 168\]](#)

[MySQL Consolidated Database \[page 170\]](#)

[Oracle Consolidated Database \[page 173\]](#)

[SAP IQ Consolidated Database \[page 182\]](#)

## 1.5.5 Adaptive Server Enterprise Consolidated Database

MobiLink supports Adaptive Server Enterprise consolidated databases.

### 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.
- 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 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 mlsvr17 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 `%SQLANY17%\MobiLink\Setup`.
- Check and update the MobiLink system setup from SQL Central.

## 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.

## Adaptive Server Enterprise considerations

### Enable functionality group configuration parameter

When the *enable functionality group* configuration parameter is enabled on the SAP Adaptive Server Enterprise 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 SAP Adaptive Server Enterprise server to avoid failure of synchronization requests.

Except for the `begin_connection_autocommit` event, events are run in chained transaction mode so that the MobiLink server can control transactions. Any called stored procedures must be defined as *chained* or *anymode* by `sp_procxmode`.

### Data type mapping

The data types of columns must map correctly between your consolidated and remote database.

### 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 `mlsrv17 -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.

### 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 `.odbc.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.

#### **Page size**

The page size of the Adaptive Server Enterprise database on which you are creating the MobiLink system tables must be 4K or greater. The MobiLink system tables cannot be created on an Adaptive Server Enterprise database with a 2K page size.

## **Related Information**

[Required Privileges for MobiLink Server \[page 23\]](#)

[MobiLink Server System Tables \[page 156\]](#)

[MobiLink System Setup](#)

[MobiLink Isolation Levels \[page 149\]](#)

[Adaptive Server Enterprise Data Mapping \[page 663\]](#)

[-b mlsrv17 Option \[page 51\]](#)

## **1.5.6 IBM DB2 LUW Consolidated Database**

MobiLink supports IBM DB2 LUW for Linux and Windows.

### **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.
- 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).

### **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.

- 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.
  - You can also check and update the MobiLink system setup from SQL Central.
- Customize `syncdb2.sql` with your connection information:
  - Copy `syncdb2.sql` to a new location where it can be modified and stored.
  - The `syncdb2.sql` script contains a default connection statement, `connect to DB2Database`

```
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.). Alter this line to connect to your IBM DB2 database. Use the following syntax:

- Run `syncdb2.sql`:

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

## ODBC Driver

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

## 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.

. Alter this line to connelf 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.

### 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 mlsrv17 -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.

### Tablespace capacity

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

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

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

For an example using the sample application, refer to the CustDB sample for MobiLink.

### 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'.

### Restrictions for tables created with a column store

Any tables involved in synchronization with MobiLink must be created with a row store if the synchronization logic requires timestamp-based downloads. This is because when tables are created with a column store, queries cannot be requested to be blocked by any uncommitted transactions for tables. This restriction would prevent the MobiLink server from providing a timestamp-based synchronization for any tables with a column store, in order to maintain data consistency.

For synchronization logic with snapshot-based downloads, synchronization tables can be created either with a column store or a row store, but the MobiLink server must be started with command line option -hwp- (which sets the MobiLink server to skip requesting the blocking behavior for generating a download for any client). Otherwise, the DB2 10.5 database server issues an error when the MobiLink server tries to generate a download for the client.

## Related Information

[Required Privileges for MobiLink Server \[page 23\]](#)

[MobiLink Server System Tables \[page 156\]](#)

[MobiLink System Setup](#)

[CustDB Sample for MobiLink](#)

[MobiLink Isolation Levels \[page 149\]](#)

[IBM DB2 LUW Data Mapping \(Deprecated\) \[page 671\]](#)

[-b mlsrv17 Option \[page 51\]](#)

## 1.5.7 Microsoft SQL Server and Microsoft Azure Consolidated Databases

MobiLink supports Microsoft SQL Server and Microsoft Azure consolidated databases.

### 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.
- 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 MobiLink server login ID must have VIEW SERVER STATE permission. This permission can be granted using the following SQL statement within the master database of a Microsoft SQL server:

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

- The MobiLink server login ID must have SELECT privilege on sys.databases SYS.DM\_TRAN\_LOCKS, SYS.PARTITIONS, and SYS.SYSPROCESSES.

### Setting up Microsoft SQL Server or Microsoft Azure as a Consolidated Database

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

- Run the `syncmss.sql` setup script, located in `%SQLANY17%\MobiLink\Setup`.
- (Microsoft SQL Server only) Check and update the MobiLink system setup from SQL Central.

### ODBC Driver

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



## Microsoft SQL Server Considerations

### Data type mapping

The data types of columns must map correctly between your consolidated and remote database.

### BLOB columns

Due to restrictions of the Microsoft SQL Server ODBC driver, place all BLOB columns at the end of the list of columns when you define a synchronization table, 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.

### 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. Use VARCHAR in the consolidated database rather than CHAR. If you must use CHAR, the `mlsrv17 -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.

### SET NOCOUNT ON

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

### Procedure calls

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

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

### Sample database uses computer columns

The Microsoft 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 Microsoft SQL Server, you must perform conflict detection and resolution in the `upload_update` script.

### Transaction durability

When using the transaction durability feature in Microsoft SQL Server 2014, the `DELAYED_DURABILITY` database option should never be set to `FORCED` because the MobiLink server requires its transactions to be fully durable.

## Considerations when using Microsoft Azure as a Consolidated Database

The following differences exist between Microsoft Azure consolidated databases and Microsoft SQL Server consolidated databases:

- To set up Microsoft Azure to work as a MobiLink consolidated database, run the setup procedure that adds MobiLink system tables, stored procedures, triggers, and views that are required for MobiLink synchronization. Run the `syncmss.sql` setup script, located in `%SQLANY17%\MobiLink\Setup`.

- The recommended ODBC driver for Microsoft Azure is ODBC Driver 13 for SQL Server. You can download this ODBC driver can be downloaded from the Microsoft download site. The other SQL Server ODBC drivers are not tested and are not recommended.
- In most cases, MobiLink supports the same functionality for Microsoft Azure as it does for Microsoft SQL Server. The following exceptions apply:

#### **MobiLink plug-in**

The MobiLink plug-in for SQL Central does not support Microsoft Azure consolidated databases.

#### **Blocking behavior**

By default, uncommitted operations (inserts, updates, and deletes) do not prevent other connections from accessing the same tables in Microsoft Azure. This behavior may differ from Microsoft SQL Server. Because of this behavior, do not use the -dsd option (disable snapshot isolation for download) when you use timestamp-based download logic for your synchronization. Doing so can cause data inconsistencies.

#### **Column default values**

Default values support only literals and constants in Microsoft Azure. Non-deterministic expressions or functions, such as GETDATE or CURRENT\_TIMESTAMP, are not supported.

## **Related Information**

[Required Privileges for MobiLink Server \[page 23\]](#)

[MobiLink Server System Tables \[page 156\]](#)

[MobiLink System Setup](#)

[Conflict Resolution with upload\\_update Scripts \[page 139\]](#)

[MobiLink Isolation Levels \[page 149\]](#)

[Recommended ODBC Drivers For MobiLink](#)

[Microsoft SQL Server Data Mapping \[page 679\]](#)

[-b mlsrv17 Option \[page 51\]](#)

## **1.5.8 MySQL Consolidated Database**

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

### **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.

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

## 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.

```
set storage_engine = [engine_name]
//
```

## Setting up MySQL as a Consolidated Database

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

- Using the MySQL command line tool or the MySQL Query Browser, run the `syncmys.sql` setup script, located in `%SQLANY17%\MobiLink\Setup`.
- Check and update the MobiLink system setup from SQL Central.

## 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 5.1.6 or later.

To specify your ODBC configuration file in UNIX or Linux, 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.

### Stored procedures prior to version 5.6.20

Versions of MySQL prior to 5.6.20 do not support the use of 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.

### Stored procedures in version 5.6.20 or later

INOUT parameters are supported in stored procedure calls. Output parameters for synchronization scripts that are written in stored procedure calls can be retrieved through INOUT parameters or returned by a result set.

### Named parameters

User-defined named parameters are not supported with MySQL prior to version 5.6.20.

### 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.

### Fractional part of a second

MySQL 5.6.20 and later supports a fractional part of a second. The number of digits in the fractional part can be a value between 0 and 6. The MySQL server rounds the fractional second into its nearest integer that can be stored into the column properly. For instance, if a timestamp column was defined as `timestamp(3)`, the server rounds 4.1234 seconds to 4.123 seconds and 4.1235 seconds to 4.124 before storing the value into the table.

In order to keep data consistency for a timestamp based download, the MobiLink server automatically subtracts one second from the `next_last_download_timestamp` generated in the `prepare_for_download` transaction before sending it to the client, if there is no user-defined `generate_next_last_download_timestamp` connection script.

## Related Information

[Required Privileges for MobiLink Server \[page 23\]](#)

[MobiLink Server System Tables \[page 156\]](#)

[MobiLink System Setup](#)

[MobiLink Isolation Levels \[page 149\]](#)

[Recommended ODBC Drivers For MobiLink](#)

[MySQL Data Mapping \[page 686\]](#)

## 1.5.9 Oracle Consolidated Database

MobiLink supports Oracle consolidated databases.

### 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.
- 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`, `V$SESSION`, `GV$LOCK`, `DBA_OBJECTS`, and `EXECUTE` privileges on `DBMS_UTILITY`. You cannot grant permission directly for the `GV$TRANSACTION`, `GV$SESSION`, `V$SESSION` and `GV$LOCK` synonyms; you must instead grant permission on the underlying `GV_$TRANSACTION`, `GV_$SESSION`, `V_$SESSION`, and `GV_$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.V_$SESSION to user-name;
```

```
grant select on SYS.GV_$LOCK to user-name;
```

```
grant select on SYS.DBA_OBJECTS 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 `%SQLANY17%\MobiLink\Setup`.
- Check and update the MobiLink system setup from SQL Central.

## ODBC Driver

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

## 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 Microsoft Windows Server 2003, the Microsoft 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 Microsoft Windows Time Service is running, to actively maintain clock synchronization. However, CTSS and Microsoft 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.

#### **XMLTYPE data type**

Use of the Oracle XMLTYPE data type with SQL Anywhere or UltraLite requires special care.

#### **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 `mlsrv17 -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.

#### **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.

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

#### **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.

#### In this section:

##### [Oracle XMLTYPE Data Type \[page 176\]](#)

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.

##### [Oracle VARRAY \[page 178\]](#)

The SQL Anywhere 17 - Oracle ODBC driver supports the use of Oracle VARRAY in stored procedures.

## Related Information

[Required Privileges for MobiLink Server \[page 23\]](#)

[SQL Anywhere 17 - Oracle ODBC Driver \[page 710\]](#)

[MobiLink System Setup](#)

[MobiLink Isolation Levels \[page 149\]](#)

[Recommended ODBC Drivers For MobiLink](#)

[Oracle Data Mapping \[page 691\]](#)

[-b mlsrv17 Option \[page 51\]](#)

### 1.5.9.1 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:

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

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

```
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**

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:

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

Then the upload\_insert script can be written as follows:

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

### **i 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:

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

## **Related Information**

[Required Privileges for MobiLink Server \[page 23\]](#)

[Oracle Data Mapping \[page 691\]](#)

## 1.5.9.2 Oracle VARRAY

The SQL Anywhere 17 - 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 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.

### Example

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;
```

## Related Information

[-s mlsrv17 Option \[page 81\]](#)

## 1.5.10 SAP HANA Consolidated Database

MobiLink supports SAP HANA consolidated databases.

### 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 access the MobiLink system tables during synchronization. This user must be used to start the MobiLink server and to configure MobiLink applications.
- The database 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 database 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 or SAP HANA cockpit. The script file is located in `%SQLANY17%\MobiLink\Setup`.

### ODBC Driver

The MobiLink server must use the SAP HANA ODBC driver from SAP that is called HDBODBC.

## SAP HANA Considerations

### MobiLink System Database (MLSD)

The MobiLink server does not support MLSD for SAP HANA.

### MobiLink server system objects

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.

### 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.

```
create COLUMN table test (pk int primary key, c1 int, last_modified
timestamp generated always as CURRENT_UTCTIMESTAMP )
```

The clause *generated always as CURRENT\_UTCTIMESTAMP* causes the SAP HANA server to update the `last_modified` column with the current UTC timestamp, whenever this row is inserted or updated.

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

```
select pk,c1 from test where last_modified > {ml s.last_table_download}
```

Similarly, the `download_delete_cursor` can be implemented by querying a deletion-tracking table populated from a delete trigger on the base table.

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.

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 log in to the SAP HANA database, then you need to grant *CATALOG READ* privilege to the MobiLink server login user using the following statement:

```
GRANT CATALOG READ TO synchronization_server_login_user_name
```

This permission 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 databases.

## Related Information

[Required Privileges for MobiLink Server \[page 23\]](#)

[MobiLink Server System Tables \[page 156\]](#)

[MobiLink System Setup](#)

[MobiLink Isolation Levels \[page 149\]](#)

[Recommended ODBC Drivers For MobiLink !\[\]\(74d4806277d7e73349d8e8c0897931e9\_img.jpg\)](#)

[SAP HANA Data Mapping \[page 701\]](#)

## 1.5.11 SQL Anywhere Consolidated Database

MobiLink supports SQL Anywhere consolidated databases.

### 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.
- 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`).

### 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 `%SQLANY17%\MobiLink\Setup`.
- Check and update the MobiLink system setup from SQL Central.

### 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.

## Related Information

[Required Privileges for MobiLink Server \[page 23\]](#)

[MobiLink Server System Tables \[page 156\]](#)

[MobiLink System Setup](#)

[ODBC Data Sources](#)

[MobiLink Isolation Levels \[page 149\]](#)

## 1.5.12 SAP IQ Consolidated Database

MobiLink supports SAP IQ consolidated databases.

### 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.
- 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 MobiLink server login ID must have EXECUTE privilege on the SP\_IQTRANSACTION system procedure for SAP IQ.

### Setting up SAP IQ as a Consolidated Database

To set up SAP 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 `%SQLANY17%\MobiLink\Setup`.
- Check and update the MobiLink system setup from SQL Central.

### Setting up the ODBC Driver

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

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

## SAP IQ Considerations

### With row-level versioning

- With the SAP 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 IQ 15.4

- If uploads contain upload data that modifies any synchronization tables that were defined on the SAP 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 IQ server allows only a single connection to modify a given table on the SAP 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:

```
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 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:

```
create table coordinate_upload ( c1 int )
```

The table is not required to have any data.

- All transactions that modify SAP IQ tables must be serialized, whether they occur on MobiLink server connections or on other connections to the SAP 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 IQ consolidated database, the table mappings default to download-only for SAP IQ tables. If you change any mappings to bi-directional or upload-only, you must ensure that changes to those SAP IQ tables are serialized. For example, by adding a `begin_upload` event as described above.

## Related Information

[Required Privileges for MobiLink Server \[page 23\]](#)

[MobiLink Server System Tables \[page 156\]](#)

[MobiLink System Setup](#)

[MobiLink Isolation Levels \[page 149\]](#)

## 1.6 MobiLink Performance

The following is a list of suggestions to help you get the best performance out of MobiLink.

## In this section:

### [Test to Improve Performance \[page 186\]](#)

The following all contribute to the throughput of your synchronization system:

### [Avoid Contention \[page 186\]](#)

Avoid contention and maximize concurrency in your synchronization scripts.

### [Use Multithreaded Network Processing \[page 187\]](#)

The MobiLink server supports multiple network worker threads processing its network streams concurrently.

### [Use an Optimal Number of Database Worker Threads \[page 187\]](#)

You can either choose a fixed number of MobiLink database worker threads or let the MobiLink server automatically adjust the number.

### [Automatic Adjustment of Database Worker Threads \[page 187\]](#)

The MobiLink server can automatically adjust the number of database worker threads based on load, with the goal of maximizing throughput.

### [Use Smaller Upload Transactions \[page 188\]](#)

Smaller uploads reduce blocking and contention, and may significantly improve throughput.

### [Avoid Synchronizing Unnecessary BLOBs \[page 188\]](#)

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 the Maximum Number of Database Connections \[page 188\]](#)

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 `mlsrv17 -cn` option.

### [Have Enough Physical Memory \[page 188\]](#)

Ensure that the computer running the MobiLink server has enough physical memory to accommodate the cache in addition to its other memory requirements.

### [Use Enough Processing Power \[page 189\]](#)

You should dedicate enough processing power to MobiLink so that the MobiLink server processing is not a bottleneck.

### [Optimize Script Execution \[page 189\]](#)

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.

### [Use Minimum Logging Verbosity \[page 189\]](#)

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.

### [Plan for Operating System Limitations \[page 189\]](#)

Operating systems restrict the number of concurrent connections a server can support over TCP/IP.

### [Java or .NET vs. SQL Synchronization Logic \[page 190\]](#)



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.

#### [Priority Synchronization \[page 190\]](#)

If you have some tables that you need to synchronize more frequently than others, create a separate publication and subscription for them.

#### [Download Only the Rows You Need \[page 191\]](#)

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 \[page 191\]](#)

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 \[page 191\]](#)

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.

#### [Use Background Synchronization \[page 191\]](#)

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 \[page 191\]](#)

The overall performance of any system, including throughput for MobiLink synchronization, is usually limited by a bottleneck at one point in the system.

#### [MobiLink Performance Monitoring \[page 195\]](#)

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

## Related Information

[Contention \[page 193\]](#)

[Transactions in the Synchronization Process](#)

[Number of Database Worker Threads \[page 193\]](#)

[MobiLink Database Connections \[page 195\]](#)

[MobiLink Profiler \[page 247\]](#)

[MobiLink Replay Utility \(mlreplay\) \[page 651\]](#)

[-wn mlsrv17 Option \[page 98\]](#)

[-wm mlsrv17 Option \[page 97\]](#)

[-w mlsrv17 Option \[page 96\]](#)

[-wu mlsrv17 Option \[page 98\]](#)

[-tu dbmlsync Option](#)

[Increment \(inc\) Extended Option](#)

[-tx mlsrv17 Option \[page 89\]](#)

[-cn mlsrv17 Option \[page 55\]](#)  
[-sm mlsrv17 Option \[page 85\]](#)  
[-nc mlsrv17 Option \[page 66\]](#)  
Communication error  
[-urc dbmlsync Option](#)  
[-cinit mlsrv17 Option \[page 54\]](#)

## 1.6.1 Test to Improve Performance

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 (mlreplay) can help with such testing.

During this testing you should experiment with the following performance tips.

## 1.6.2 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.

## 1.6.3 Use Multithreaded 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 mlsrv17` option to set stream threads.

## 1.6.4 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.

## 1.6.5 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 mlsrv17` option to set the initial number of concurrent database worker threads and the `-wm mlsrv17` 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 mlsrv17` 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.

## 1.6.6 Use Smaller Upload Transactions

Smaller uploads reduce blocking and contention, and may significantly improve throughput.

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.

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.
- 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.

On the server side, the performance can be tuned by using the `-tx mlsrv17` 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.

Test and tune these client-side and server-side options for maximum throughput.

## 1.6.7 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.

## 1.6.8 Set the 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 `mlsrv17 -cn` option.

## 1.6.9 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.

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 swapping to disk, its throughput will fall significantly so it is very important that the MobiLink server has physical memory to process these synchronizations.

## 1.6.10 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.

## 1.6.11 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 SQL Central to create your MobiLink applications, an index is automatically defined for each download cursor when you deploy the model.

## 1.6.12 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.

## 1.6.13 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 mlsrv17` option, or use the `-sm mlsrv17` 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.

## 1.6.14 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.

## 1.6.15 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 SQL 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.

## 1.6.16 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.

## 1.6.17 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.

## 1.6.18 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.

## 1.6.19 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.

## 1.6.20 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.

#### **In this section:**

##### [MobiLink Tuning for Performance \[page 192\]](#)

The key to achieving optimal MobiLink synchronization throughput is to have multiple synchronizations occurring simultaneously and executing efficiently.

## **1.6.20.1 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.

#### **In this section:**

##### [Contention \[page 193\]](#)

The most important factor is to avoid database contention in your synchronization scripts.

##### [Number of Database Worker Threads \[page 193\]](#)

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.

##### [MobiLink Database Connections \[page 195\]](#)

MobiLink creates a database connection for each database worker thread. You can use the `-cn` option to specify that MobiLink create a larger pool of database connections, but any excess connections are idle unless MobiLink needs to close a connection or use a different script version.



## 1.6.20.1.1 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.

## Related Information

[-sm mlsrv17 Option \[page 85\]](#)

[SendDownloadAck \(sa\) Extended Option](#)

## 1.6.20.1.2 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 mlsrv17` option to set the number of database worker threads. You can also use the `-wm mlsrv17` 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.

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.

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.

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.

## Related Information

[-w mlsrv17 Option \[page 96\]](#)

[-wm mlsrv17 Option \[page 97\]](#)

[-wu mlsrv17 Option \[page 98\]](#)

### 1.6.20.1.3 MobiLink Database Connections

MobiLink creates a database connection for each database worker thread. You can use the `-cn` option to specify that MobiLink create a larger pool of database connections, but any excess connections are idle unless MobiLink needs to close a connection or use a different script version.

There are two cases where MobiLink closes a database connection and open a new one. The first case is if an error occurs. The second case is if the client requests a synchronization script version, and none of the available connections have already used that synchronization script version.

#### i Note

Each database connection is associated with a script version. To change the script version, the connection must be closed and reopened.

If you routinely use more than one script version, you can reduce the need for MobiLink to close and open connections by increasing the number of connections. You can eliminate the need completely if the number of connections used for synchronizations is the number of database worker threads times the number of script versions.

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

```
mksrv17 -c "DSN=SQL Anywhere 17 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.

## Related Information

[-cn mksrv17 Option \[page 55\]](#)

## 1.6.21 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.

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.

SAP Solution Manager provides tools that you can use to monitor MobiLink as part of your overall SAP landscape. The `ncs.conf` file provides all the necessary connection and configuration information required to deliver monitoring data to an SAP Diagnostic Agent. To enable monitoring, use the `-ncs`, `-ncsd`, or `-ncsp` MobiLink server options.

## Related Information

[MobiLink Profiler \[page 247\]](#)

[MobiLink Server Logging and SAP Passports \[page 26\]](#)

[download\\_statistics Connection Event \[page 398\]](#)

[download\\_statistics Table Event \[page 401\]](#)

[synchronization\\_statistics Connection Event \[page 487\]](#)

[synchronization\\_statistics Table Event \[page 491\]](#)

[time\\_statistics Connection Event \[page 494\]](#)

[time\\_statistics Table Event \[page 497\]](#)

[upload\\_statistics Connection Event \[page 513\]](#)

[upload\\_statistics Table Event \[page 517\]](#)

[-ncs mlsrv17 Option \[page 67\]](#)

[-ncsd mlsrv17 Option \[page 68\]](#)

[-ncsp mlsrv17 Option \[page 69\]](#)

## 1.7 MobiLink Client/Server Communications Encryption

You can encrypt MobiLink client/server communication using transport layer security.

### In this section:

[End-to-end Encryption \[page 196\]](#)

End-to-end encryption occurs when data is encrypted at the point of origin and decrypted at the final destination.

[Starting the MobiLink Server with Transport Layer Security \[page 197\]](#)

To start the MobiLink server with transport layer security, supply the identity file and the identity password protecting the server's private key.

[MobiLink Client Configuration to Use Transport Layer Security \[page 198\]](#)

You can configure SQL Anywhere or UltraLite clients to use MobiLink transport layer security.

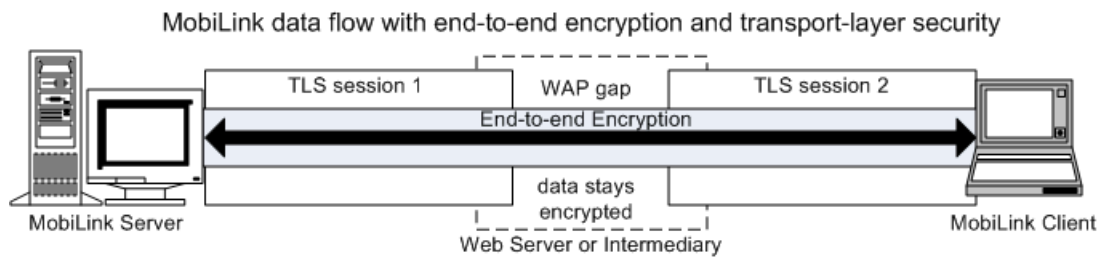
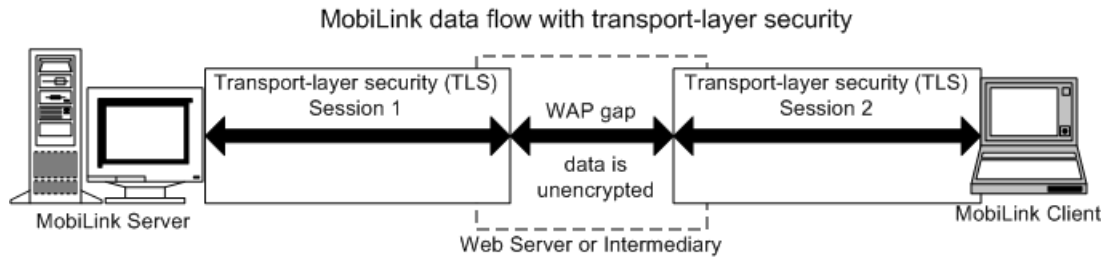
### 1.7.1 End-to-end Encryption

End-to-end encryption occurs when data is encrypted at the point of origin and decrypted at the final destination.

There is no point during transmission that the data is unencrypted.

In MobiLink, transport layer security (TLS) is sometimes only used to encrypt data up to an intermediary (for example, encryption/decryption hardware) between the client and server. At the intermediary, the data would be decrypted and then encrypted again by the intermediary for the rest of the journey. Notably, this happens when synchronizing via HTTPS through a Web server. The brief interval when the data is unencrypted in the intermediary is sometimes called the Wireless Application Protocol gap or WAP Gap.

Within a corporation, a WAP gap is often acceptable when the intermediary is within corporate control. However, in a third-party hosted environment where data from different corporations is going through the same WAP gap, sensitive data may be exposed. End-to-end encryption prevents any intermediary from accessing the data because the synchronization stream is encrypted from start to finish, and may optionally be encrypted once more with TLS.



## 1.7.2 Starting the MobiLink Server with Transport Layer Security

To start the MobiLink server with transport layer security, supply the identity file and the identity password protecting the server's private key.

### Context

You can hide the command line options using a configuration file and the File Hiding utility (dbfhide).

### Procedure

1. Use the `mksrv17 -x` server option to specify an identity and an identity password. The syntax for specifying secure communications options is:

```
-x protocol (
```

```
FIPS={ y | n };  
IDENTITY=identity-file;  
IDENTITY_PASSWORD=password;... )
```

2. Set the options as follows:

#### **protocol**

The protocol to use. It can be *https* or *tls*. The *tls* protocol indicates the use of TCP/IP with Transport Layer Security.

#### **FIPS**

Indicates whether to use FIPS 140-2 certified RSA encryption. Servers using the FIPS option are compatible with clients not using the FIPS option and vice versa.

#### **identity-file**

The path and file name of the identity file, which contains the server's private key, the server's certificate, and, optionally, the certificates signed by the Certificate Authority.

#### **password**

The password for the server private key. You specify this password when you create the server identity.

## **Results**

You have started the MobiLink server with transport layer security.

## **Example**

The following example specifies transport layer security, the server identity file, and the identity password protecting the server's private key.

```
mksrv17 -c "dsn=my_cons"  
-x tls(identity=rsaserver.id;identity_password=test)
```

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

```
mksrv17 -c "dsn=my_cons"  
-x "tls(identity=C:\Users\Public\Documents\SQL Anywhere  
17\Samples\Certificates\rsaserver.id;identity_password=pwd) "
```

## **1.7.3 MobiLink Client Configuration to Use Transport Layer Security**

You can configure SQL Anywhere or UltraLite clients to use MobiLink transport layer security.

For each client, you specify trusted certificates, the type of encryption, and the network protocol.

## In this section:

### [Server Authentication \[page 199\]](#)

Server authentication allows a remote client to verify the identity of a MobiLink server.

### [Client Security Options \[page 200\]](#)

MobiLink clients (SQL Anywhere and UltraLite) use a common set of connection parameters to configure transport layer security.

### [Transport Layer Security over TCP/IP and HTTPS \[page 201\]](#)

MobiLink transport layer security is an inherent feature of the MobiLink HTTPS and TCP/IP protocols.

## 1.7.3.1 Server Authentication

Server authentication allows a remote client to verify the identity of a MobiLink server.

Digital signatures and certificate field verification work together to achieve server authentication.

### Digital Signatures

A MobiLink server certificate contains one or more digital signatures used to maintain data integrity and protect against tampering. Following are the steps used to create a digital signature:

- An algorithm performed on a certificate generates a unique value or hash.
- The hash is encrypted using a signing certificate's or Certificate Authority's private key.
- The encrypted hash, called a digital signature, is embedded in the certificate.

A digital signature can be self-signed or signed by an enterprise root certificate or Certificate Authority.

When a MobiLink client contacts a MobiLink server, and each is configured to use transport layer security, the server sends the client a copy of its certificate. The client decrypts the certificate's digital signature using the server's public key included in the certificate, calculates a new hash of the certificate, and compares the two values. If the values match, this confirms the integrity of the server's certificate.

### Verifying Certificate Fields

When using a globally signed certificate, each client must verify certificate field values to avoid trusting certificates that the same Certificate Authority has signed for other clients. This is resolved by requiring your clients to test the value of fields in the identity portion of the certificate. A Certificate Authority must guarantee the accuracy of the identification information in any certificate that it signs.

When creating a certificate using the createcert utility, you enter values for the organization, organizational unit, and common name fields. You verify these fields using corresponding MobiLink client connection parameters.

#### Organization

The organization field corresponds to the certificate\_company MobiLink client connection parameter.

### **Organizational unit**

The organizational unit field corresponds to the certificate\_unit MobiLink client connection parameter.

### **Common name**

The common name field corresponds to the certificate\_name MobiLink client connection parameter.

## **Related Information**

[Self-signed Root Certificates](#)

[Certificate Chains](#)

[Globally Signed Certificates](#)

[Client Security Options \[page 200\]](#)

[Digital Certificates](#)

[How to Set up Transport Layer Security](#)

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

[certificate\\_company MobiLink Client Network Protocol Option](#)

[certificate\\_unit MobiLink Client Network Protocol Option](#)

[certificate\\_name MobiLink Client Network Protocol Option](#)

## **1.7.3.2 Client Security Options**

MobiLink clients (SQL Anywhere and UltraLite) use a common set of connection parameters to configure transport layer security.

### **trusted\_certificates Protocol Option**

MobiLink clients use the trusted\_certificates protocol option to specify trusted MobiLink server certificates. The trusted certificate can be a server's self-signed certificate, a public root certificate, or the certificate belonging to a commercial Certificate Authority. The trusted\_certificates protocol option accepts either name of the file that contains the certificate or the certificate itself as a PEM-encoded string.

### **Verifying Certificate Fields**

The certificate\_company, certificate\_unit, and certificate\_name protocol options are used to verify certificate fields, an important step for server authentication. Verify certificate fields if you are using a third-party Certificate Authority to globally sign certificates.



## Related Information

[Digital Certificates](#)

[Globally Signed Certificates](#)

[Server Authentication \[page 199\]](#)

[trusted\\_certificates](#) [MobiLink Client Network Protocol Option](#)

### 1.7.3.3 Transport Layer Security over TCP/IP and HTTPS

MobiLink transport layer security is an inherent feature of the MobiLink HTTPS and TCP/IP protocols.

To use transport layer security over HTTPS, specify the TRUSTED\_CERTIFICATES connection parameter using the ADR extended option. Following is the syntax for a partial dbmlsync command line.

```
-e "ctp=protocol;  
  adr=[ FIPS={ y | n }; ]  
      TRUSTED_CERTIFICATES={ public-certificate-filename | PEM-encoded-certificate };  
  ..."
```

#### **protocol**

The protocol to use. It can be *https* or *tls*. The *tls* (transport layer security) protocol uses RSA encryption over TCP/IP.

#### **FIPS**

Indicates whether to use FIPS-certified encryption. FIPS-certified encryption can only be used with RSA encryption. FIPS-certified HTTPS uses separate FIPS 140-2 certified software, but is compatible with version 9.0.2 or later MobiLink servers using HTTPS.

#### **public-certificate-filename**

The path and file name of a trusted certificate.

For HTTPS or FIPS-certified HTTPS, you must use certificates created using RSA encryption.

#### **PEM-encoded-certificate**

A string that contains the PEM-encoded certificate.

For HTTPS or FIPS-certified HTTPS, you must use certificates created using RSA encryption.

For TLS and HTTPS synchronizations, if none of the `certificate_name`, `certificate_company`, or `certificate_unit` protocol options are provided, MobiLink clients check the host name against the certificate provided by the server. If the Subject Alternative Name extension is present, the client checks the host name against each of the names in the extension. Otherwise, the client checks the host name against the Common Name in the subject field. If no matches are found, the synchronization fails with `STREAM_ERROR_SECURE_CERTIFICATE_NOT_TRUSTED`. This check supports wildcard matching.

If any of the `certificate_name`, `certificate_company`, or `certificate_unit` protocol options are provided, these values are used to check against the server certificate's subject field, and the host name will be ignored. If the `skip_certificate_name_check` protocol option is enabled, no name checking will be done on the server's certificate.

## Example

The following example specifies RSA security over HTTPS. In this example, the client checks that the name(s) on the server's certificate matches the host name "myserver". The synchronization fails if the host name does not match. It must all be written on one line:

```
dbmlsync -c "server=rem1;uid=DBA;pwd=mypwd"
-e "ctp=https;
    adr='host=MyServer;
    trusted_certificates=c:\certs\rootca.crt'"
```

Alternatively, you can specify the CommunicationAddress extended option using the CREATE SYNCHRONIZATION SUBSCRIPTION or ALTER SYNCHRONIZATION SUBSCRIPTION statement. This method provides the same information, but stores it in the database.

```
CREATE SYNCHRONIZATION SUBSCRIPTION
TO pub1
FOR user1
ADDRESS 'host=MyServer;
    trusted_certificates=c:\certs\rootca.crt';
```

In the following example, the host name matching is overridden with a direct check of three of the subject fields of the server's certificate. It must all be written on one line:

```
dbmlsync -c "server=rem1;uid=myuid;pwd=mypwd"
-e "ctp=https;
    adr='host=myserver;
    trusted_certificates=c:\certs\rootca.crt;
    certificate_company=My Company;
    certificate_unit=My Division;
    certificate_name=My MobiLink Server'"
```

## 1.8 Manage Remote Databases

You can centrally manage remote databases involved in MobiLink synchronization using the MobiLink plug-in SQL Central.

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.

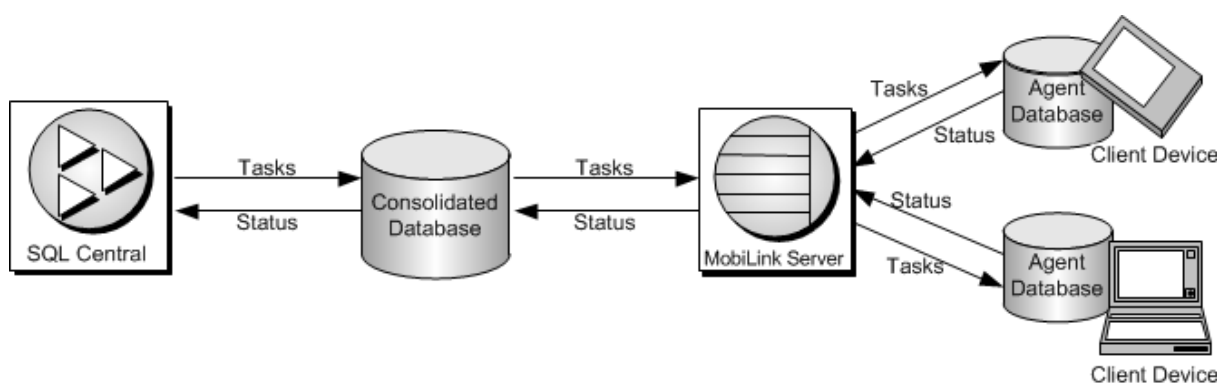
Central administration is achieved using remote tasks. A remote task is an ordered collection of commands (similar to a batch file) that you create using the MobiLink 17 plug-in for SQL Central. A remote task may have conditions that govern when it runs. A remote task can be configured to run only once, 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 17 plug-in for SQL Central.

The following diagram shows how information flows between SQL Central, the consolidated database, and the client devices when using the central administration of remote databases feature.



#### In this section:

##### [Central Administration Concepts \[page 204\]](#)

The following concepts are important to understand when working with central administration of remote databases.

##### [MobiLink Agents \[page 207\]](#)

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.

##### [Remote Tasks \[page 219\]](#)

A remote task is the unit of work when performing central administration of remote databases.

##### [Deployment and Configuration \[page 245\]](#)

Information is provided about deployment and configuration. For a list of files required to deploy the MobiLink Agent, see the SQL Anywhere MobiLink client deployment and Deploying UltraLite MobiLink clients topics.

## 1.8.1 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 17 plug-in for SQL 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 `%SQLANYSAMPI7%\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 must 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 startup with the `magent` command or a default value is assigned in the form `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.

## 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 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 17\diagnostics\MobiLink Agent on Microsoft Windows and My Device\Application Data\SQLAny17\MLAgent on Microsoft 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 SQL 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.

### In this section:

[Central Administration Setup Overview \[page 206\]](#)

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

### 1.8.1.1 Central Administration 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 SQL Central.
2. Use SQL Central to define one or more remote schema names to identify your remote database(s) to the system.
3. Use SQL Central to make all the Agents that are managing the remote databases known to your system.

4. Use SQL Central to create tasks and assign them to Agents.

#### **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.

## **Related Information**

[Remote Tasks \[page 219\]](#)

[MobiLink Agent on the Client Device \[page 209\]](#)

[MobiLink Agent on the Client Device \[page 209\]](#)

[Creating a MobiLink Project](#)

[Adding a Remote Schema Name \[page 214\]](#)

[Adding an Agent \[page 215\]](#)

## **1.8.2 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.

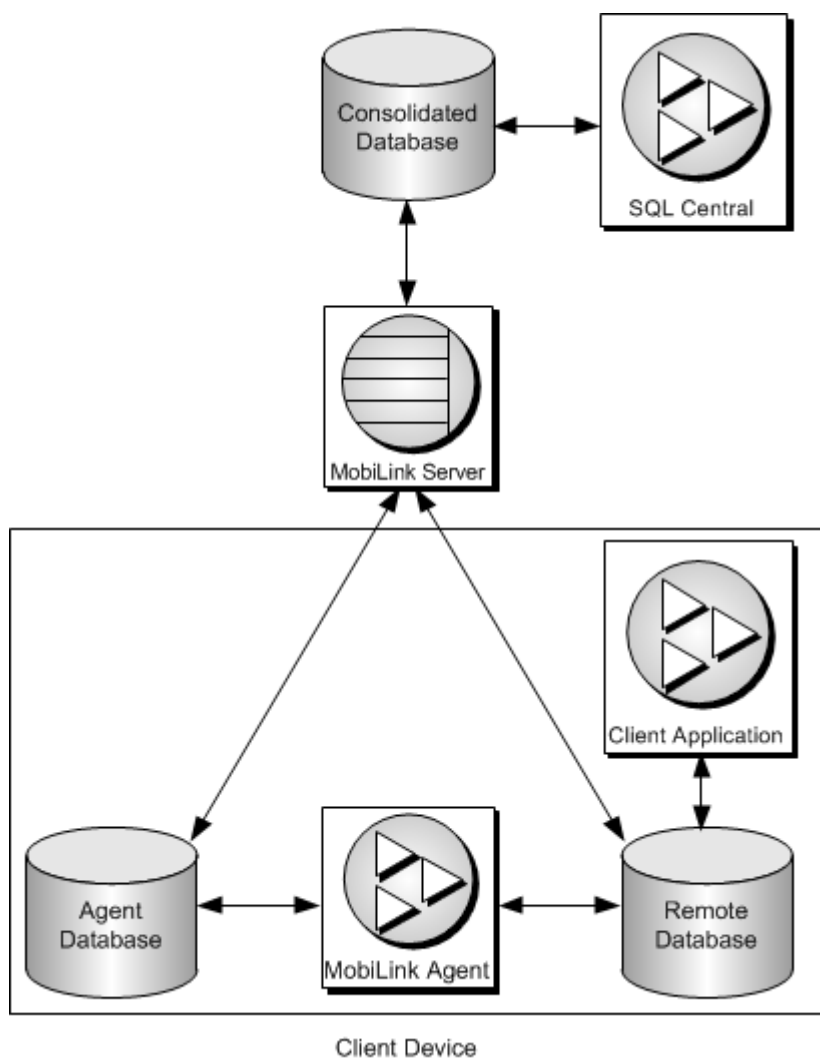
The Agent synchronizes the agent database using ordinary MobiLink synchronization. During synchronization, the Agent receives new tasks to execute and uploads information about tasks it has executed.

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 multithreaded 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.



**In this section:**

[MobiLink Agent on the Client Device \[page 209\]](#)

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.

[MobiLink Agents in SQL Central \[page 213\]](#)

After an Agent is created and configured on the client device using the `mlagent` command, the Agent must also be created in SQL Central before it can be assigned tasks.

[Agent Authentication \[page 218\]](#)

The Agent acts as a MobiLink synchronization client when synchronizing the agent database.



## 1.8.2.1 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 continues 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.

### In this section:

#### [mlagent Command \[page 209\]](#)

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

#### [Interactive Configuration of the MobiLink Agent \[page 212\]](#)

The MobiLink Agent can be configured using a configuration window as well as through the command line.

#### [MobiLink Agent Stop Utility \[page 213\]](#)

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.

### 1.8.2.1.1 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.

Option	Description
<code>@ data</code>	Use this to read in options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used.
<code>-c</code>	Set the configuration options, only updating those options that differ from the current options, then stop the Agent.

Option	Description
<code>-cr</code>	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.
<code>-a agentid</code>	Valid only with <code>-c</code> or <code>-cr</code> . Specify the ID of this Agent.  If the <code>-a</code> option is not specified, the default is <code>Agent_computername_UUID</code> , where <code>computername</code> is the host name of the computer that the Agent is running on and <code>UUID</code> 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 <code>Agent_XYZ</code> and <code>agent_xyz</code> 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.
<code>-db database location</code>	Valid only with <code>-c</code> or <code>-cr</code> . Specify the path where remote databases are stored on the client device.
<code>-X protocol[protocol-options]...</code> <code>protocol: tcpip   tls   http   https</code> <code>protocol-options: (option=value; ...)</code>	Valid only with <code>-c</code> or <code>-cr</code> . Specify the MobiLink client network protocol options. These options determine how the Agent connects to the MobiLink server when synchronizing the agent database.
<code>-ap parameters</code>	Specify the MobiLink authentication parameters used when synchronizing the agent database.
<code>-ek key</code>	Specify the encryption key used to access the agent database.
<code>-n name</code>	Specify the name of the agent database. The default is <code>taskdb</code> .
<code>-o file</code>	Log output to the specified file.
<code>-on size</code>	Append <code>.old</code> to the <code>mlagent</code> log file name and start a new file with the original name when the log reaches the <code>size</code> specified. Size must be a minimum of 10K. This option cannot be used with <code>-os</code> .
<code>-os size</code>	Rename the <code>mlagent</code> log file to <code>YYMMDDxx.ml.a</code> and start a new file with the original name when the log reaches the <code>size</code> specified. Size must be a minimum of 10K. This option cannot be used with <code>-on</code> .

Option	Description
<code>-ot file</code>	Truncate the file and log output messages to it.
<code>-p password</code>	Specify the MobiLink password used to synchronize the agent database.
<code>-pi</code>	<p>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 the <code>-pi dbmlsync</code> option.</p> <p>The MobiLink Agent database must be configured by running mlagent with <code>-c</code> or <code>-cr</code> before mlagent is invoked with <code>-pi</code>.</p> <p>The MobiLink Agent cannot be invoked with both <code>-pi</code> and either <code>-c</code> or <code>-cr</code> on the command line.</p>
<code>-q</code>	Run in a minimized window.
<code>-qi</code>	Do not display tray icon or window.
<code>-u user</code>	Specify the MobiLink user name used to synchronize the agent database.
<code>-v level</code>	Specify the output verbosity level from 0-9. The default is 3.

## Example

The following example demonstrates how to run the Agent in configuration mode. It uses the default Agent database and sets the agent ID to be the same as the MobiLink user ID:

```
mlagent -c -a username -u username -p password -x
http{host=myhost.example.com;port=8080} -o logfile.mla
```

The following example demonstrates how to run the Agent in normal mode, accepting all the settings from the default agent database:

```
mlagent
```

## Related Information

[Configuration Files](#)  
[-pi dbmlsync Option](#)

### 1.8.2.1.2 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 Microsoft Windows, click the System menu in the top left corner of the *MobiLink Agent* window and click *Configure*. For Microsoft 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.

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

## 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, a 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, a window appears so you can enter the key.

## 1.8.2.1.3 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	
<code>mlastop [ options ]</code>	
Option	Description
<code>-n name</code>	The name of the Agent to stop. If -n is not specified, all Agents on the device are stopped.

### Related Information

[mlagent Command \[page 209\]](#)

[SQL Anywhere MobiLink Client Deployment \[page 18\]](#)

[UltraLite MobiLink Client Deployment \[page 21\]](#)

## 1.8.2.2 MobiLink Agents in SQL Central

After an Agent is created and configured on the client device using the `mlagent` command, the Agent must also be created in SQL 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 SQL Central.

### In this section:

[Adding a Remote Schema Name \[page 214\]](#)

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

[Importing Remote Schema Names \[page 214\]](#)

Schema names can be imported from another database.

[Adding an Agent \[page 215\]](#)

Add an Agent to use central administration of remote databases.

[Agent Properties \[page 216\]](#)

Agent properties can be viewed and edited from the *Agent Properties* window in SQL Central.

[Adding Managed Remote Databases \[page 217\]](#)

In order for an Agent to manage a remote database, the database must be associated with the Agent in SQL Central. This is done by specifying a remote schema name.

[Adding a Group \[page 217\]](#)

A group is a collection of MobiLink Agents.

### 1.8.2.2.1 Adding a Remote Schema Name

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

#### Context

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

#### Procedure

1. Double-click the MobiLink project.
2. Double-click *Remote Schema Names* and then click **► New ► Remote Schema Name ►**.
3. Follow the instructions in the *Create Remote Schema Name Wizard* and click *Finish*.

#### Results

The remote schema name is added.

### 1.8.2.2.2 Importing Remote Schema Names

Schema names can be imported from another database.

#### Procedure

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.

### 1.8.2.2.3 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 SQL Central.

## Procedure

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 Steps

Once the Agent is created in SQL Central, you can do the following:

- view and change Agent properties
- add remote databases for an Agent to manage
- synchronize an Agent
- delete an Agent
- view events for an Agent
- view tasks for an Agent
- view information about remote databases managed by an Agent

## 1.8.2.2.4 Agent Properties

Agent properties can be viewed and edited from the [Agent Properties](#) window in SQL 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 SQL 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).

#### **In this section:**

[Viewing or Changing Agent Properties \[page 216\]](#)

Agent properties can be viewed or updated.

### 1.8.2.2.4.1 Viewing or Changing Agent Properties

Agent properties can be viewed or updated.

#### **Procedure**

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

### 1.8.2.2.5 Adding Managed Remote Databases

In order for an Agent to manage a remote database, the database must be associated with the Agent in SQL Central. This is done by specifying a remote schema name.

#### Procedure

1. Double-click the consolidated database.
2. Double-click the Agent you want to work 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.

### 1.8.2.2.6 Adding a Group

A group is a collection of MobiLink Agents.

#### Prerequisites

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

#### Procedure

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.

### 1.8.2.3 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_17` 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.

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_17', 'authenticate_user', '<DEFINE  
YOUR AUTHENTICATION LOGIC>' )
```

- Use *Connection Scripts* in SQL Central. You can use the `ml_ra_agent_17` 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.

## Related Information

[Consolidated Database Setup \[page 155\]](#)

[Script Versions \[page 312\]](#)

[authenticate\\_user Connection Event \[page 354\]](#)

[authenticate\\_user\\_hashed Connection Event \[page 360\]](#)

[authenticate\\_parameters Connection Event \[page 351\]](#)

## 1.8.3 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 17 plug-in for SQL Central. At design-time they are stored locally on the administrator's computer in the project.

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 17 plug-in for SQL 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.

#### **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.

### In this section:

#### [Remote Task Logic \[page 222\]](#)

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

#### [Creating a Remote Task \[page 223\]](#)

Tasks are defined and managed within the context of a MobiLink project.

#### [Editing a Remote Task \[page 224\]](#)

Remote task properties can be edited.

#### [Deploying a Remote Task \[page 225\]](#)

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

#### [Exporting a Remote Task \[page 226\]](#)

Remote tasks can be exported to a file.

### [Deployed Remote Tasks \[page 226\]](#)

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.

### [Server-initiated Remote Tasks \(SIRT\) \[page 230\]](#)

A server-initiated remote task is any remote task that is run when the Agent receives notification from the server to run the task.

### [Task Commands \[page 232\]](#)

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.

### [Variables in Parameters \[page 241\]](#)

The following macros provide access to information that may vary between remote devices.

### [Status \[page 242\]](#)

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.

### [MobiLink System Procedures \[page 243\]](#)

In addition to the functionality in SQL Central for managing remote tasks, there are also MobiLink system procedures in the consolidated database that can be used to automate administration tasks.

## 1.8.3.1 Remote Task Logic

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

```
current_command = 1;
num_tries = 0;
EXECUTE_TASK:
loop {
    num_tries = num_tries + 1;
    EXECUTE_COMMANDS;
    if( task_success or task_abort ) break EXECUTE_TASK;
    if( task_retry and at maximum tries ) {
        break EXECUTE_TASK;
    } else {
        continue;
    }
}
EXECUTE_COMMANDS:
for each command starting at current_command {
    execute current_command;
    if( command_failed ) {
        if( action_on_failure is "abort task" ) {
            break EXECUTE_COMMANDS, returning task_abort;
        } else if( action_on_failure is "continue" ) {
            // no action, continue at next command
        } else if( action_on_failure is "retry task" ) {
            current_command = 1;
            break EXECUTE_COMMANDS, returning task_retry;
        } else if( action_on_failure is "retry command" ) {
            // no change to current_command
            break EXECUTE_COMMANDS, returning task_retry;
        }
    }
}
current_command = next command number;
```

```
}  
return task_success;
```

The maximum running time is reset for a task when the task starts and when a command or the entire task is retried because a command fails.

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.

## 1.8.3.2 Creating a Remote Task

Tasks are defined and managed within the context of a MobiLink project.

### Context

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.

### Procedure

1. Double-click the MobiLink project.
2. Double-click *Remote Tasks* and click **New** > *Remote Task*.
3. Follow the steps in the *Create Remote Task Wizard* and click *Finish*.

#### i Note

If you must 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* stays 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.

### Results

The task is created.

## Next Steps

After the task has been created, it can be deployed to the consolidated, and then assigned to recipients (Agents).

## Related Information

[Adding a Command to a Remote Task \[page 240\]](#)

[Deploying a Remote Task \[page 225\]](#)

### 1.8.3.3 Editing a Remote Task

Remote task properties can be edited.

#### Procedure

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.

## Next Steps

A remote task can be deployed and assigned to Agents.



## 1.8.3.4 Deploying a Remote Task

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

### Prerequisites

A remote task must contain at least one command before it can be deployed.

### Context

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.

### Procedure

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 Steps

Deployed tasks can be canceled, initiated, reactivated and have recipients added to them.

### Related Information

[Task Commands \[page 232\]](#)

[Adding a Command to a Remote Task \[page 240\]](#)

[Deployed Remote Tasks \[page 226\]](#)

## 1.8.3.5 Exporting a Remote Task

Remote tasks can be exported to a file.

### Prerequisites

You must have a remote task defined.

### Procedure

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.

## 1.8.3.6 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 SQL 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 17 plug-in for SQL 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.

**In this section:**

#### [Canceling a Deployed Remote Task for All Agents \[page 227\]](#)

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.

#### [Canceling a Deployed Remote Task for a Single Agent \[page 228\]](#)

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.

#### [Initiating a Deployed Remote Task for All Agents \[page 228\]](#)

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

#### [Initiating a Deployed Remote Task for a Single Agent \[page 229\]](#)

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

#### [Reactivating a Deployed Remote Task for a Single Agent \[page 229\]](#)

Some deployed remote tasks can be reactivated.

#### [Adding Recipients to a Deployed Remote Task \[page 230\]](#)

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

## 1.8.3.6.1 Canceling a Deployed Remote Task for All Agents

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.

### Procedure

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.

## 1.8.3.6.2 Canceling a Deployed Remote Task for a Single Agent

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.

### Procedure

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

The deployed remote task is canceled.

## 1.8.3.6.3 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.

### Procedure

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.

### 1.8.3.6.4 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.

#### Procedure

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.

### 1.8.3.6.5 Reactivating a Deployed Remote Task for a Single Agent

Some deployed remote tasks can be reactivated.

#### Procedure

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 reactivate and click *Reactivate*.

#### Results

The deployed task is reactivated.

## 1.8.3.6.6 Adding Recipients to a Deployed Remote Task

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

### Procedure

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.

## 1.8.3.7 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 SQL 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.

### In this section:

[Remote Task Notifier \(RTNotifier\) \[page 231\]](#)

A notifier called the RTNotifier is built into the MobiLink server to keep track of SIRT requests.

### Related Information

[ml\\_ra\\_notify\\_task System Procedure \[page 642\]](#)

## 1.8.3.7.1 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.

```
call ml_add_property( 'SIRT', 'RTNotifier(RTNotifier1)', 'enable', 'yes' );
```

The table below lists RTNotifier options that can be specified in the ml\_property system table.

Option	Value	Description
autoset_poll_every	{ <i>yes</i>   <i>no</i> }	Specifies whether the poll_every property should be automatically adjusted based on how often the Agents poll the MobiLink server for remote task requests. If the poll_every is adjusted, the update is seen when the RTNotifier checks for updates based on the update_poll_every value.
enable	{ <i>yes</i>   <i>no</i> }	Specifies whether the RTNotifier should be enabled when the MobiLink server starts. This property cannot be updated dynamically.
poll_every	time in seconds	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.
update_poll_every	time in seconds	Specifies how often, in seconds, the RTNotifier should check for updates to their properties.

Option	Value	Description
request_cursor		This option is for internal use only. Specifies a query used to retrieve the remote task requests from the consolidated database.

## 1.8.3.8 Task 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.

### In this section:

#### [Copy File Command \[page 233\]](#)

Makes a copy of a file on the remote device.

#### [Create Database Command \[page 233\]](#)

Creates a new remote database on the remote device that is managed by the Agent.

#### [Delete File Command \[page 234\]](#)

Deletes a file on the remote device.

#### [Download File Command \[page 235\]](#)

Download a file from the server to the remote device.

#### [Drop Database Command \[page 235\]](#)

Deletes a managed remote database.

#### [Execute SQL Command \[page 236\]](#)

Executes SQL against a remote database.

#### [Prompt Command \[page 237\]](#)

Displays a message box on the remote device.

#### [Rename File Command \[page 237\]](#)

Renames a file on the remote device.

#### [Run Program Command \[page 238\]](#)

Run a program on the remote device.

#### [Synchronize Command \[page 238\]](#)

Synchronizes a remote database.

#### [Upload File Command \[page 239\]](#)

Uploads a file from the remote device to the server.

#### [Command Usage \[page 239\]](#)

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



## 1.8.3.8.1 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.

## 1.8.3.8.2 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. You should 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 Microsoft Windows Mobile.

The CREATE DATABASE statement can be used to initialize a database on a desktop computer, which can later be copied to a Microsoft Windows Mobile device.

## **1.8.3.8.3 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 17 plug-in for SQL Central.

## 1.8.3.8.4 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.

#### 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.

### Related Information

[-ftr mlsrv17 Option \[page 64\]](#)

## 1.8.3.8.5 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 Microsoft Windows Mobile.

## Related Information

[UltraLite DBF Connection Parameter](#)

[DatabaseFile \(DBF\) Connection Parameter](#)

### 1.8.3.8.6 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:

```
SELECT * FROM systable
GO
CREATE PROCEDURE p1()
BEGIN
    CREATE TABLE t1( pk INTEGER PRIMARY KEY );
    INSERT INTO t1 VALUES( 5 );
    COMMIT;
END
GO
SELECT * FROM SYSPROCEDURE
```

#### Remarks

This command can only be used in a remote task marked as requiring or creating a remote database. The SQL is executed against the database associated with the remote schema name specified for the remote task.

When executing SQL, the Agent does not COMMIT any statements. If the SQL being execute does not have a COMMIT, the statement is rolled back. This is important when the SQL is INSERT, UPDATE and DELETE statements or any other statements that do not explicitly cause a COMMIT.

The status /results for the command store the results of the executed SQL. DDL statements return no results. INSERT/UPDATE/DELETE statements return the number of rows affected as a single value on a line. SELECT statements return the results in .CSV format with column headings as the first row. Results from multiple statements are all appended into one big result.

## 1.8.3.8.7 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.

## 1.8.3.8.8 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.

### 1.8.3.8.9 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.

### 1.8.3.8.10 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.

### 1.8.3.8.11 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`.

## Related Information

[Variables in Parameters \[page 241\]](#)

### 1.8.3.8.12 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*.

**In this section:**

[Adding a Command to a Remote Task \[page 240\]](#)

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

## 1.8.3.8.12.1 Adding a Command to a Remote Task

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

### Procedure

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.
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 Steps

The task can be deployed.

## Related Information

[Task Commands \[page 232\]](#)

[Deploying a Remote Task \[page 225\]](#)

### 1.8.3.9 Variables in Parameters

The following macros provide access to information that may vary between remote devices.

These values can be used in remote task conditions, in parameters for commands within remote tasks, and in connection strings:

Variable	Replacement
{agent_db}	The full path and file name for the agent database file. This file can be uploaded from the device to help diagnose problems when necessary.
{agent_id}	The Agent ID.
{agent_log}	The full path and file name for the Agent log file. This file can be uploaded from the device to help diagnose problems when necessary. If the Agent runs without an Agent log file, this variable is an empty string.
{battery_level}	The battery level for the remote device. The range is zero to one hundred (0-100).
{db_location}	The Agent's remote database directory, as specified by the <code>m1agent -db</code> option.
{is_on_ac_power}	Indicates whether 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.
{is_online}	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).

Variable	Replacement
{ml_password}	The MobiLink password (pwd) being used by the Agent to synchronize with the agent database.
{ml_stream}	The protocol parameters for connecting to the MobiLink server. For example, HTTP {host=SAP.com;port=9376}
{ml_username}	The MobiLink user name (uid) being used by the Agent to synchronize with the agent database.
{network_conn_name}	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 ?.
{remote_id}	The remote ID for the remote database associated with this task. This value is only meaningful for in remote tasks that are marked as <i>Requiring or Creating a Remote Database</i> .
{rows_to_upload}	For UltraLite databases only. The number of rows in the remote database that will be uploaded if a full synchronization is done.

## 1.8.3.10 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 17 plug-in for SQL 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

## Related Information

[ml\\_ra\\_get\\_agent\\_events System Procedure \[page 629\]](#)  
[ml\\_ra\\_get\\_agent\\_ids System Procedure \[page 633\]](#)  
[ml\\_ra\\_get\\_agent\\_properties System Procedure \[page 634\]](#)  
[ml\\_ra\\_get\\_latest\\_event\\_id System Procedure \[page 635\]](#)  
[ml\\_ra\\_get\\_orphan\\_taskdbs System Procedure \[page 635\]](#)  
[ml\\_ra\\_get\\_remote\\_ids System Procedure \[page 636\]](#)  
[ml\\_ra\\_get\\_task\\_results System Procedure \[page 637\]](#)  
[ml\\_ra\\_get\\_task\\_status System Procedure \[page 639\]](#)

### 1.8.3.11 MobiLink System Procedures

In addition to the functionality in SQL 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 17 plug-in for SQL Central. However, the following tasks can only be done using the MobiLink 17 plug-in for SQL 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

## Related Information

[ml\\_ra\\_add\\_agent\\_id](#) System Procedure [page 623]  
[ml\\_ra\\_assign\\_task](#) System Procedure [page 623]  
[ml\\_ra\\_cancel\\_notification](#) System Procedure [page 624]  
[ml\\_ra\\_cancel\\_task\\_instance](#) System Procedure [page 625]  
[ml\\_ra\\_clone\\_agent\\_properties](#) System Procedure [page 626]  
[ml\\_ra\\_delete\\_agent\\_id](#) System Procedure [page 627]  
[ml\\_ra\\_delete\\_events\\_before](#) System Procedure [page 627]  
[ml\\_ra\\_delete\\_remote\\_id](#) System Procedure [page 628]  
[ml\\_ra\\_delete\\_task](#) System Procedure [page 629]  
[ml\\_ra\\_get\\_agent\\_events](#) System Procedure [page 629]  
[ml\\_ra\\_get\\_agent\\_ids](#) System Procedure [page 633]  
[ml\\_ra\\_get\\_agent\\_properties](#) System Procedure [page 634]  
[ml\\_ra\\_get\\_latest\\_event\\_id](#) System Procedure [page 635]  
[ml\\_ra\\_get\\_orphan\\_taskdbs](#) System Procedure [page 635]  
[ml\\_ra\\_get\\_remote\\_ids](#) System Procedure [page 636]  
[ml\\_ra\\_get\\_task\\_results](#) System Procedure [page 637]  
[ml\\_ra\\_get\\_task\\_status](#) System Procedure [page 639]  
[ml\\_ra\\_manage\\_remote\\_db](#) System Procedure [page 641]  
[ml\\_ra\\_notify\\_agent\\_sync](#) System Procedure [page 642]  
[ml\\_ra\\_reassign\\_taskdb](#) System Procedure [page 643]  
[ml\\_ra\\_set\\_agent\\_property](#) System Procedure [page 644]  
[ml\\_ra\\_unmanage\\_remote\\_db](#) System Procedure [page 645]

## 1.8.4 Deployment and Configuration

Information is provided about deployment and configuration. For a list of files required to deploy the MobiLink Agent, see the SQL Anywhere MobiLink client deployment and Deploying UltraLite MobiLink clients topics.

### 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 Microsoft Windows and Microsoft 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.
- 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.

### Deployment of UltraLite Applications and Databases with the MobiLink Agent on Microsoft Windows Mobile

There are various mechanisms to deploy UltraLite with the Agent on a Microsoft 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 Microsoft 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.

### Deployment 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*.

## Configuration Considerations for the Agent (Microsoft Windows desktop)

One way to configure the Agent on a Microsoft 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 `magent -cr` on the command line.

After the agent is configured, the install program could run `magent -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 `magent` failed to ping the MobiLink server, the return code from `magent.exe` would be the SQLCODE that results from a synchronization of the Agent database with the configured `magent` options.

## Configuration Considerations for the Agent (Microsoft Windows Mobile)

How you configure the MobiLink Agent on Microsoft 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 `magent` 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.

## Related Information

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

[MobiLink Agent on the Client Device \[page 209\]](#)

[SQL Anywhere MobiLink Client Deployment \[page 18\]](#)

[UltraLite MobiLink Client Deployment \[page 21\]](#)

[MobiLink Agent Stop Utility \[page 213\]](#)

[magent Command \[page 209\]](#)

## 1.9 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.

### i Note

Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were previously performed in the Monitor can be performed in the SQL Anywhere Cockpit. See [SQL Anywhere Monitor Non-GUI User Guide](#).

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.

### In this section:

[Starting the MobiLink Profiler \(Administration Tools\) \[page 249\]](#)

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.

[MobiLink Profiler \(mlprof\) on the Command Line \[page 250\]](#)

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.

#### [Starting a Profiling Session \[page 250\]](#)

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

#### [Ending a Profiling Session \[page 252\]](#)

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.

#### [Opening or Deleting a Previous Profiling Session \[page 253\]](#)

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

#### [The Profiling Database \[page 253\]](#)

When the MobiLink Profiler first starts, it creates a profiling database with a default file name, user and password.

#### [MobiLink Profiler Interface \[page 254\]](#)

The MobiLink Profiler has the following panes.

#### [Statistic Customization \[page 263\]](#)

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.

#### [Using the Profiling Database \[page 264\]](#)

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

#### [MobiLink Synchronization Statistical Properties \[page 267\]](#)

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.

## Related Information

[SQL Anywhere Monitor](#)



## 1.9.1 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

#### i Note

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

### Procedure

Click **Start > Programs > SQL Anywhere 17 > Administration Tools > MobiLink Profiler**.

### Results

The MobiLink Profiler is started.

### Next Steps

Begin a profiling session to start collecting data.

### Related Information

[Starting a Profiling Session \[page 250\]](#)

## 1.9.2 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:

```
mlprof [ options ]
```

Option	Description
<code>-c</code>	Closes the MobiLink Profiler at the end of the profiling session.
<code>-p password</code>	The password for the MobiLink user.
<code>-r</code>	Recreates the profiling database. Use this option to remove all previous profiling sessions, or if there is a problem with the profiling database schema.
<code>-U ml_username</code>	The MobiLink user. This option is required to begin a profiling session initiated from the command line.
<code>-x {tcpip tls http https}[( keyword=value;... )]</code>	The network protocol and parameters for connecting to the MobiLink server. The keyword=value pairs can be the host, port, and additional network parameters. This option is required to begin a profiling session initiated from the command line.

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

### Related Information

[-x mlsrv17 Option \[page 99\]](#)

## 1.9.3 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.

## Procedure

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.

### 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.

3. Start synchronizing.

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.

## Related Information

[MobiLink Client/Server Communications Encryption \[page 196\]](#)

[MobiLink Client Network Protocol Options](#)

## 1.9.4 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.

## Procedure

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.

## 1.9.5 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.

### Procedure

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 Steps

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

## 1.9.6 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 `MLProfiler17` 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.




## 1.9.7 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.

### 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* .

### 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* .

### 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.

In addition, there is an *Options* window that you can use to customize the display and property windows that can be displayed.

### In this section:

#### [Details Table Pane \[page 255\]](#)

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.

#### [Utilization Graph Pane \[page 257\]](#)

The *Utilization Graph* is the second pane from the top. It displays the number of synchronizations in each phase in a time graph.

#### [Chart Pane \[page 258\]](#)

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.

#### [Overview Pane \[page 260\]](#)

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.

#### Options Window [page 261]

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.

#### Session Properties [page 262]

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 [page 262]

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.

#### Synchronization Properties [page 262]

Double-click a synchronization in either the *Details Table* pane or the *Chart* pane to see properties for that synchronization.

## 1.9.7.1 *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.

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.

### **remote\_id**

The unique identifier of the remote database.

### **user**

The MobiLink synchronization user.

### **version**

The version of the synchronization script.

### **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.

**end\_sync 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.

**get\_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.

## **Related Information**

[MobiLink Synchronization Statistical Properties \[page 267\]](#)

[Synchronization Properties \[page 262\]](#)

[Remote IDs](#)

[MobiLink Users](#)

[Script Versions \[page 312\]](#)

## **1.9.7.2 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.

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.

### **In this section:**

[How the Utilization Graph Works \[page 258\]](#)

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.

## Related Information

[Marquee Tool \[page 260\]](#)

[Sample Properties \[page 262\]](#)

### 1.9.7.2.1 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.

### 1.9.7.3 *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.

## 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.

## 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.

Use the *Options* window to change color settings.

## Related Information

[Synchronization Properties \[page 262\]](#)

[Marquee Tool \[page 260\]](#)

[Options Window \[page 261\]](#)

### 1.9.7.4 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*.

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.

#### In this section:

[Marquee Tool \[page 260\]](#)

The *Marquee Tool* is the small box that appears in the *Overview* pane. You can use it 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:

## Related Information

[Options Window \[page 261\]](#)

[Statistic Customization \[page 263\]](#)

### 1.9.7.4.1 Marquee Tool

The *Marquee Tool* is the small box that appears in the *Overview* pane. You can use it 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.

**In this section:**

[Changing the Color of the Marquee Tool \[page 261\]](#)

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

### 1.9.7.4.1.1 Changing the Color of the *Marquee Tool*

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

#### Procedure

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.

## 1.9.7.5 *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**.

## 1.9.7.6 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**.

## 1.9.7.7 Sample Properties

The *Sample Properties* window provides detailed statistics for time intervals. Each time interval is about one second long. Samples are numbered by the MobiLink Profiler to reflect the order in which they were received.

To open the *Sample Properties* window, click in the *Graph* pane for the time period that you want to examine.

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.

## 1.9.7.8 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*.

## Related Information

[MobiLink Synchronization Statistical Properties \[page 267\]](#)

### 1.9.8 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*

**In this section:**

[Creating a New Watch \[page 264\]](#)

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

## 1.9.8.1 Creating a New Watch

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

### Procedure

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*.
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.

### Related Information

[MobiLink Synchronization Statistical Properties \[page 267\]](#)

## 1.9.9 Using the Profiling Database

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

### Prerequisites

#### i Note

Adobe will stop updating and distributing the Flash Player at the end of 2020. Because the SQL Anywhere Monitor is based on Flash, you cannot use it once Flash support ends. In many cases, tasks that were previously performed in the Monitor can be performed in the SQL Anywhere Cockpit. See [SQL Anywhere Monitor Non-GUI User Guide](#).



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

This task assumes you are using the default profiling database.

You can also use Interactive SQL to work with Profiler views.

## Procedure

1. Use the [SQL Anywhere 17](#) 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 `MLProfiler17` 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:

```
dbeng17.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\\_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)
- [server\\_throughput\\_samples](#)
- [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.

#### **i** 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.

## Example

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

```
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:

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

## Next Steps

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

## 1.9.10 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.

### Synchronization Statistics

MobiLink statistical properties return the following information for synchronizations:

Property	Description
<i>active</i>	(hidden by default) True if the synchronization is active as of the time you viewed the information for this synchronization.
<i>apply_upload</i>	(phase counter) Time required for the uploaded data to be applied to the consolidated database.
<i>authenticate_user</i>	(phase counter) Total time to perform user authentication, including executing the <code>authenticate_*</code> events.
<i>begin_sync</i>	(phase counter) Total time for the <code>begin_synchronization</code> event.
<i>client</i>	(hidden by default) The type of MobiLink client and full client version. For example, <code>dbmlsync 16.0.0.xxxx</code> .
<i>completed</i>	(hidden by default) True if the synchronization completed successfully.
<i>conflicted_updates</i>	(hidden by default) Number of update rows that caused conflict. A row is included only when a resolve conflict script was successfully called for it.
<i>connect</i>	(phase counter) 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.
<i>connect_for_download_ack</i>	(phase counter) Time required by the database worker thread to make a database connection if a new database connection is needed for a download acknowledgement.
<i>connection_retries</i>	(hidden by default) Number of times the MobiLink server re-tried the connection to the consolidated database.

Property	Description
<i>download</i>	(hidden by default) This property indicates the synchronization included a download command.
<i>download_ack</i>	(hidden by default) Can be none or non-blocking.
<i>download_bytes</i>	(hidden by default) Amount of memory used within the MobiLink server to store the download and send to the remote database (before any encryption or compression).
<i>download_deleted_rows</i>	(hidden by default) Number of row deletions fetched from the consolidated database by the MobiLink server (using <code>download_delete_cursor</code> scripts).
<i>download_errors</i>	(hidden by default) Number of errors that occurred during the download.
<i>download_fetched_rows</i>	(hidden by default) Number of rows fetched from the consolidated database by the MobiLink server (using <code>download_cursor</code> scripts).
<i>download_filtered_rows</i>	(hidden by default) Number of fetched rows that were not downloaded to the MobiLink client because they matched rows that the client uploaded.
<i>download_warnings</i>	(hidden by default) Number of warnings that occurred during the download.
<i>duration</i>	Total time for the synchronization, as measured by the MobiLink server.
<i>end_sync</i>	(phase counter) Total time for the end_synchronization event.
<i>fetch_download</i>	(phase counter) Time required to fetch the rows to be downloaded from the consolidated database to create the download stream.
<i>get_db_worker</i>	(phase counter) Time required to acquire a free database worker thread.
<i>get_db_worker_for_download_ack</i>	(phase counter) Time spent waiting for a free database worker thread after the download acknowledgement has been received.
<i>has_blocked</i>	(hidden by default) True if blocking is detected by the MobiLink server.
<i>ignored_deletes</i>	Number of upload delete rows that caused errors while the <code>upload_delete</code> script was invoked, when the <code>handle_error</code> or <code>handle_odbc_error</code> are defined and returned 1000, or when there is no <code>upload_delete</code> script defined for the given table.
<i>ignored_inserts</i>	Total number of upload insert rows that were ignored. They were ignored because 1) there is no <code>upload_insert</code> script in normal mode; or 2) errors occurred when the MobiLink server was invoking the corresponding script and the <code>handle_error</code> or <code>handle_odbc_error</code> event returned 1000.
<i>ignored_updates</i>	Number of upload update rows that caused conflict but a resolve conflict script was not successfully called or no <code>upload_update</code> script was defined.

Property	Description
<i>nonblocking_download_ack</i>	(phase counter) Time required for the publication_nonblocking_download_ack connection and nonblocking_download_ack connection events.
<i>number</i>	Synchronization number.
<i>prepare_for_download</i>	(phase counter) Total time for the prepare_for_download event.
<i>receive_upload</i>	(phase counter) Phase time for receiving the upload.
<i>remote_id</i>	Remote ID that uniquely identifies the remote database.
<i>send_download</i>	(phase counter) 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.
<i>server</i>	(hidden by default) The MobiLink server name or host:port.
<i>start_time</i>	Date-time (in ISO-8601 extended format) for the start of the synchronization.
<i>sync_deadlocks</i>	(hidden by default) Number of deadlocks in the consolidated database that were detected for the synchronization.
<i>sync_errors</i>	(hidden by default) Total number of errors that occurred for the synchronization.
<i>sync_request</i>	(phase counter) 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.
<i>sync_tables</i>	(hidden by default) Number of client tables that were involved in the synchronization.
<i>sync_warnings</i>	(hidden by default) Number of warnings that occurred for the synchronization.
<i>upload</i>	(hidden by default) Indicates the synchronization included an upload command.
<i>upload_bytes</i>	(hidden by default) 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.
<i>upload_deadlocks</i>	(hidden by default) Number of deadlocks in the consolidated database that were detected during the upload.
<i>upload_deleted_rows</i>	(hidden by default) Number of rows that were successfully deleted from the consolidated database.
<i>upload_errors</i>	(hidden by default) Number of errors that occurred during the upload.
<i>upload_inserted_rows</i>	(hidden by default) Number of rows that were successfully inserted in the consolidated database.
<i>upload_updated_rows</i>	(hidden by default) Number of rows that were successfully updated in the consolidated database.
<i>upload_warnings</i>	Number of warnings that occurred during the upload.

Property	Description
<i>user</i>	MobiLink user name.
<i>version</i>	Name of the synchronization version.
<i>wait_for_download_ack</i>	(phase counter) Time spent waiting for the download to be applied to the remote database and for the remote database to send the download acknowledgement.

## Related Information

[Statistic Customization \[page 263\]](#)

[Details Table Pane \[page 255\]](#)

[Synchronization Properties \[page 262\]](#)

## 1.10 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 SAP 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 transaction log file, which otherwise grows large and slows down synchronization. A full synchronization may also be required to recover from errors.

### In this section:

#### [File-Based Download Setup \[page 271\]](#)

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.

#### [Validation Checks \[page 275\]](#)

Before applying a download file to a remote database, dbmlsync does several things to ensure that the synchronization is valid.

#### [File-Based Download Examples \[page 278\]](#)

The following two examples set 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.

## Related Information

[Download-only Publications](#)

### 1.10.1 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.
2. At the consolidated database, create scripts with a new script version.
3. Create a download file.
4. Apply the download file.

### In this section:

#### [File-Definition Database \[page 272\]](#)

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.

#### [Changes at the Consolidated Database \[page 272\]](#)

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.

#### [Download File Creation \[page 273\]](#)

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 `dbmsync` with the `-bc` option and supply a file name with the extension `.df`.

#### [Application of the Download File \[page 274\]](#)

If you need 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 `dbmsync -bg` option when creating the download file.

## Related Information

### [File-Based Download Examples \[page 278\]](#)

### 1.10.1.1 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.

### 1.10.1.2 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.

### i 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.

## Related Information

[Script Versions \[page 312\]](#)

[begin\\_publication Connection Event \[page 374\]](#)

[modify\\_last\\_download\\_timestamp Connection Event \[page 457\]](#)

[-dsd mlsrv17 Option \[page 59\]](#)

### 1.10.1.3 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=passwd;SERVER=fbd1_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.

## Related Information

[-be dbmsync Option](#)

[sp\\_hook\\_dbmsync\\_validate\\_download\\_file](#)

[-bg dbmsync Option](#)

### 1.10.1.4 Application of the Download File

If you need 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 dbmsync -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 dbmsync 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.

## Related Information

[MobiLink Generation Numbers \[page 277\]](#)

[-bg dbmlsync Option](#)

### 1.10.2 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.
- Optionally, `dbmlsync` checks the generation number in the remote database to ensure it matches the generation number in the download file.
- 

**In this section:**

### [Automatic Validation \[page 276\]](#)

Before applying a download file, dbmsync performs special checks on the last download timestamp, next last download timestamp, download file creation time, and transaction log.

### [MobiLink Generation Numbers \[page 277\]](#)

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.

### [Custom Validation \[page 278\]](#)

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_dbmsync_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.

## 1.10.2.1 Automatic Validation

Before applying a download file, dbmsync 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 dbmsync 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, dbmsync scans the remote database's transaction log and builds up a list of all changes that must be uploaded. Dbmsync only applies a download file if it does not contain any operations that affect rows with changes that must be uploaded.

### 1.10.2.2 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.

#### **i** 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_dbmsync_validate_download_file` stored procedure can be used to override the default checking of the generation number.

## Related Information

[begin\\_publication Connection Event \[page 374\]](#)

[end\\_publication Connection Event \[page 415\]](#)

[sp\\_hook\\_dbmsync\\_validate\\_download\\_file](#)

### 1.10.2.3 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_dbmsync_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 `dbmsync -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_dbmsync_validate_download_file` through the `#hook_dict` table, and can be used in your validation logic.

## Related Information

[sp\\_hook\\_dbmsync\\_validate\\_download\\_file](#)

### 1.10.3 File-Based Download Examples

The following two examples set 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.

#### In this section:

##### [Snapshot Example \[page 279\]](#)

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.

##### [Timestamp-Based Example \[page 283\]](#)

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.

### 1.10.3.1 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,passwd scon.s.db
dbinit -dba DBA,passwd sremote.db
dbinit -dba DBA,passwd 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.

```
dbeng17 -n sfdef_eng sfdef.db
dbeng17 -n scon_eng scon.s.db
dbeng17 -n sremote_eng sremote.db
dbdsn -y -w fbd_demo -c
"SERVER=scon_eng;DBF=scon.s.db;UID=DBA;PWD=passwd;ASTART=off;ASTOP=off"
```

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

```
read "C:\Program Files\SQL Anywhere 17\MobiLink\setup\syncsa.sql"
```

Start the MobiLink server:

```
start mlsrv17 -v+ -c "DSN=fbd_demo" -zu+ -ot scon.txt
```

#### Set up the Snapshot Example Consolidated Database

In this example, the consolidated database has one table, called T1. After connecting to the consolidated database, you can run the following SQL to create table T1:

```
CREATE TABLE T1 (
  pk INTEGER PRIMARY KEY,
  c1 INTEGER
);
```

The following code creates a script version called `filebased` and creates a download script for that script version.

```
CALL ml_add_table_script( 'filebased',
  'T1', 'download_cursor',
  'SELECT pk, c1 FROM T1' );
```

The following code creates a script version called normal and creates upload and download scripts for that script version.

```
CALL ml_add_table_script ( 'normal', 'T1',
  'upload_insert',
  'INSERT INTO T1 VALUES ({ml r.pk}, {ml r.c1})');
CALL ml_add_table_script( 'normal', 'T1',
  'upload_update',
  'UPDATE T1 SET c1 = {ml r.c1} WHERE pk = {ml r.pk} ' );
CALL ml_add_table_script( 'normal', 'T1',
  'upload_delete',
  'DELETE FROM T1 WHERE pk = {ml r.pk}' );
CALL ml_add_table_script( 'normal', 'T1',
  'download_cursor',
  'SELECT pk, c1 FROM T1' );
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:

```
CREATE PROCEDURE begin_pub (
  INOUT generation_num integer,
  IN   username          varchar(128),
  IN   pubname           varchar(128) )
BEGIN
  SET generation_num=1;
END;
CALL ml_add_connection_script(
  'filebased',
  'begin_publication',
  '{ call begin_pub(
    {ml s.generation_number},
    {ml s.username},
    {ml s.publication_name} ) }' );
CALL ml_add_connection_script(
  'normal',
  'begin_publication',
  '{ call begin_pub(
    {ml s.generation_number},
    {ml s.username},
    {ml s.publication_name} ) }' );
```

## Create the Snapshot Example Remote Database

In this example, the remote database also contains one table, called T1. Connect to the remote database and run the following SQL to create the table T1, a publication called P1, and a user called U1. The SQL also creates a subscription for U1 to P1.

```
CREATE TABLE T1 (
  pk INTEGER PRIMARY KEY,
  c1 INTEGER
);
CREATE PUBLICATION P1 (
  TABLE T1
);
CREATE SYNCHRONIZATION USER U1;
CREATE SYNCHRONIZATION SUBSCRIPTION
```



```
TO P1  
FOR U1;
```

The following code creates an `sp_hook_dbmlsync_validate_download_file` hook to implement user-defined validation logic in the remote database:

```
CREATE PROCEDURE sp_hook_dbmlsync_validate_download_file()  
BEGIN  
    DECLARE udata varchar(256);  
    SELECT value  
        INTO udata  
        FROM #hook_dict  
        WHERE name = 'user data';  
    IF udata <> 'ok' THEN  
        UPDATE #hook_dict  
        SET value = 'FALSE'  
        WHERE name = 'apply file';  
    END IF;  
END
```

## Create the Snapshot Example File-Definition Database

A file-definition database is required in MobiLink systems that use file-based download. This database has the same schema as the remote databases being updated by file-based download, and it contains no data or state information. The file-definition database is used solely to define the structure of the data that is to be included in the download file. One file-definition database can be used for many groups of remote databases, each defined by its own MobiLink group user name.

The following code defines the file-definition database for this sample. It creates a schema that is identical to the remote database, and also creates:

- a publication called P1 that publishes all rows of the T1 table. The same publication name must be used in the file-definition database and the remote databases.
- a MobiLink user called G1. This user represents all the remotes that are to be updated in the file-based download.
- a subscription to the publication.

You must connect to `sfdef.db` before running this code.

```
CREATE TABLE T1 (  
    pk INTEGER PRIMARY KEY,  
    c1 INTEGER  
);  
CREATE PUBLICATION P1 (  
    TABLE T1  
);  
CREATE SYNCHRONIZATION USER G1;  
CREATE SYNCHRONIZATION SUBSCRIPTION  
TO P1  
FOR G1;
```

## Prepare for Initial Synchronization

To prepare your new remote database so that you can apply a download file, you need to either perform a normal synchronization or create the download file with the `dbmlsync -bg` option. This example shows you how to initialize your new remote database by performing a normal synchronization.

You can perform an initial synchronization of the remote database with the script version called `normal` that was created earlier:

```
dbmlsync -c "UID=DBA;PWD=passwd;SERVER=sremote_eng;DBF=sremote.db" -v+ -e  
"sv=normal"
```

## Demonstrate the Snapshot Example File-Based Download

Connect to the consolidated database and insert some data that is synchronized by file-based download, such as the following:

```
INSERT INTO T1 VALUES( 1, 1 );  
INSERT INTO T1 VALUES( 2, 4 );  
INSERT INTO T1 VALUES( 3, 9 );  
COMMIT;
```

The following command must be run on the computer that holds the file-definition database. It does the following:

- The `dbmlsync -bc` option creates the download file, and names it `file1.df`.
- The `-be` option includes the string "OK" in the download file that is accessible to the `sp_dbmlsync_validate_download_file` hook.

```
dbmlsync -c "UID=DBA;PWD=passwd;SERVER=sfdef_eng;DBF=sfdef.db" -v+ -e  
"sv=filebased" -bc file1.df -be ok -ot fdef.txt
```

To apply the download file, run `dbmlsync` with the `-ba` option on the remote database, supplying the name of the download file you want to apply:

```
dbmlsync -c "UID=DBA;PWD=passwd;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:

```
SELECT * FROM T1
```

## Clean up the Snapshot Example

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

```
del file1.df  
mlstop -h -w
```

```
dbstop -y -c "SERVER=sfdef_eng; UID=DBA; PWD=passwd"
dbstop -y -c "SERVER=scons_eng; UID=DBA; PWD=passwd"
dbstop -y -c "SERVER=sremote_eng; UID=DBA; PWD=passwd"
dberase -y sfdef.db
dberase -y scons.db
dberase -y sremote.db
```

### 1.10.3.2 Timestamp-Based Example

This example implements file-based download for timestamp-based synchronization. It sets up the three databases and then demonstrates how to download data by file.

This example is presented in such a way that you can either just read through it, or you can cut and paste the text to run the sample.

#### Create Databases for the Sample

The following commands create the three databases used in the example: a consolidated database, a remote database, and a file-definition database.

```
dbinit -dba DBA,passwd tcons.db
dbinit -dba DBA,passwd tremote.db
dbinit -dba DBA,passwd 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.

```
dbeng17 -n tfdef_eng tfdef.db
dbeng17 -n tcons_eng tcons.db
dbeng17 -n tremote_eng tremote.db
dbdsn -y -w tfbd_demo -c
"SERVER=tcons_eng;DBF=tcons.db;UID=DBA;PWD=passwd;START=off;ASTOP=off"
```

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

```
read "C:\Program Files\SQL Anywhere 17\MobiLink\setup\syncsa.sql"
```

Start the MobiLink server:

```
mlsrv17 -v+ -c "DSN=tfbd_demo" -zu+ -ot tcons.txt
```

#### Set up the Timestamp Example Consolidated Database

In this example, the consolidated database has one table, called T1. After connecting to the consolidated database, you can run the following code to create T1:

```
CREATE TABLE T1 (
  pk INTEGER PRIMARY KEY,
```

```

c1 INTEGER,
last_modified TIMESTAMP DEFAULT TIMESTAMP
);

```

The following code defines a script version called normal with a minimal number of scripts. This script version is used for synchronizations that do **not** use file-based download.

```

CALL ml_add_table_script( 'normal', 'T1',
  'upload_insert',
  'INSERT INTO T1( pk, c1) VALUES( {ml r.pk}, {ml r.c1} )' );

CALL ml_add_table_script( 'normal', 'T1',
  'upload_update',
  'UPDATE T1 SET c1 = {ml r.c1} WHERE pk = {ml r.pk} ' );
CALL ml_add_table_script( 'normal', 'T1',
  'upload_delete',
  'DELETE FROM T1 WHERE pk = {ml r.pk}');
CALL ml_add_table_script( 'normal', 'T1',
  'download_cursor',
  'SELECT pk, c1 FROM T1
  WHERE last_modified >= {ml s.last_table_download}' );

```

The following code sets the generation number for all subscriptions to 1. Generation numbers can be useful if your consolidated database becomes lost or corrupted and you need to force an upload.

```

CREATE PROCEDURE begin_pub (
  INOUT generation_num integer,
  IN  username          varchar(128),
  IN  pubname           varchar(128) )
BEGIN
  SET generation_num = 1;
END;

CALL ml_add_connection_script( 'normal',
  'begin_publication',
  '{ call begin_pub(
    {ml s.generation_number},
    {ml s.username},
    {ml s.publication_name},
    {ml s.last_publication_upload},
    {ml s.last_publication_download} ) }' );

COMMIT;

```

The following code defines the script version called filebased. This script version is used to create file-based download.

```

CALL ml_add_connection_script( 'filebased',
  'begin_publication',
  '{ call begin_pub(
    {ml s.generation_number},
    {ml s.username},
    {ml s.publication_name} ) }' );
CALL ml_add_table_script( 'filebased', 'T1',
  'download_cursor',
  'SELECT pk, c1 FROM T1
  WHERE last_modified >= {ml s.last_table_download}' );

```

The following code sets the last download time so that all changes that occurred within the last five days are included in download files. Any remote 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.

```

CREATE PROCEDURE ModifyLastDownloadTimestamp(

```

```

        INOUT last_download_timestamp TIMESTAMP,
        IN     ml_username             VARCHAR(128) )
BEGIN
    SELECT dateadd( day, -5, CURRENT_TIMESTAMP )
    INTO last_download_timestamp;
END;

CALL ml_add_connection_script( 'filebased',
    'modify_last_download_timestamp',
    'CALL ModifyLastDownloadTimestamp(
        {ml s.last_download}, {ml s.username} )' );

COMMIT;

```

## Create the Timestamp Example Remote Database

In this example, the remote database also contains one table, called T1. After connecting to the remote database, run the following code to create table T1, a publication called P1, and a user called U1. The code also creates a subscription for U1 to P1.

```

CREATE TABLE T1 (
    pk INTEGER PRIMARY KEY,
    c1 INTEGER
);
CREATE PUBLICATION P1 (
    TABLE T1
);
CREATE SYNCHRONIZATION USER U1;
CREATE SYNCHRONIZATION SUBSCRIPTION
TO P1
FOR U1;

```

The following code defines an `sp_hook_dbmlsync_validate_download_file` stored procedure. This stored procedure prevents the application of download files that do not have the string "ok" embedded in them.

```

CREATE PROCEDURE sp_hook_dbmlsync_validate_download_file()
BEGIN
    DECLARE udata varchar(256);

    SELECT value
    INTO udata
    FROM #hook_dict
    WHERE name = 'user data';

    IF udata <> 'ok' THEN
        UPDATE #hook_dict
        SET value = 'FALSE'
        WHERE name = 'apply file';
    END IF;
END

```

## Create the Timestamp Example File-Definition Database

The following code defines the file-definition database for the timestamp example. It creates a table, a publication, a user, and a subscription for the user to the publication.

```
CREATE TABLE T1 (  
  pk INTEGER PRIMARY KEY,  
  c1 INTEGER  
);  
CREATE PUBLICATION P1 (  
  TABLE T1  
);  
CREATE SYNCHRONIZATION USER G1;  
CREATE SYNCHRONIZATION SUBSCRIPTION  
TO P1  
FOR G1;
```

## Prepare for Initial Synchronization

To prepare your new remote database so that you can apply a download file, you need to either perform a normal synchronization or create the download file with the `dbmlsync -bg` option. This example shows you how to use `-bg`.

The following code defines a script version called `filebased_init` for the consolidated database. This script version has a single `begin_publication` script.

```
CALL ml_add_table_script(  
  'filebased_init', 'T1', 'download_cursor',  
  'SELECT pk, c1 FROM T1' );  
CALL ml_add_connection_script(  
  'filebased_init',  
  'begin_publication',  
  '{ call begin_pub(  
    {ml s.generation_number},  
    {ml s.username},  
    {ml s.publication_name} ) }' );  
  
COMMIT;
```

The following two command lines create and apply an initial download file using the script version called `filebased_init` and the `-bg` option.

```
dbmlsync -c "UID=DBA;PWD=passwd;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=passwd;SERVER=tremote_eng;DBF=tremote.db"  
-v+ -ba tfile1.df -ot tremote.txt
```

## Demonstrate the Timestamp Example File-Based Download

Connect to the consolidated database and insert some data that is synchronized by file-based download, such as the following:

```
INSERT INTO T1(pk, c1) VALUES( 1, 1 );
INSERT INTO T1(pk, c1) VALUES( 2, 4 );
INSERT INTO T1(pk, c1) VALUES( 3, 9 );
commit;
```

The following command line creates a download file containing the new data.

```
dbmlsync -c
  "UID=DBA;PWD=passwd;SERVER=tfdef_eng;DBF=tfdef.db"
  -v+ -e "sv=filebased" -bc tfile2.df -be ok -ot tfdef2.txt
```

The following command line applies the download file to the remote database.

```
dbmlsync -c "UID=DBA;PWD=passwd;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:

```
SELECT * FROM T1
```

## Clean up the Timestamp Example

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

```
del tfile1.df
mlstop -h -w
dbstop -y -c "SERVER=tfdef_eng; UID=DBA; PWD=passwd"
dbstop -y -c "SERVER=tcons_eng; UID=DBA; PWD=passwd"
dbstop -y -c "SERVER=tremote_eng; UID=DBA; PWD=passwd"
dberase -y tfdef.db
dberase -y tcons.db
dberase -y tremote.db
```

## 1.11 The Relay Server Reverse Proxy

The Relay Server is a reverse proxy that enables secure, load-balanced communication between mobile devices and backend servers through a web server. Supported backend servers include MobiLink, SAP Mobile Server, SAP Afaria, and SAP Mobile Office.

The Relay Server provides the following:

- A common communication architecture for mobile devices communicating with backend servers.
- A mechanism to enable a load-balanced and fault-tolerant environment for backend 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.

## 1.12 MobiLink Events

The synchronization process has multiple steps and a unique event identifies each step. You control the synchronization process by writing scripts associated with some of these events.

### In this section:

#### [Synchronization Scripts \[page 288\]](#)

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, or Java, or .NET.

#### [Synchronization Events \[page 332\]](#)

A MobiLink synchronization is made up of many events.

### 1.12.1 Synchronization Scripts

You control the synchronization process by writing synchronization scripts and storing or referencing them in MobiLink system tables in the consolidated database. You can write scripts in SQL, or 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)

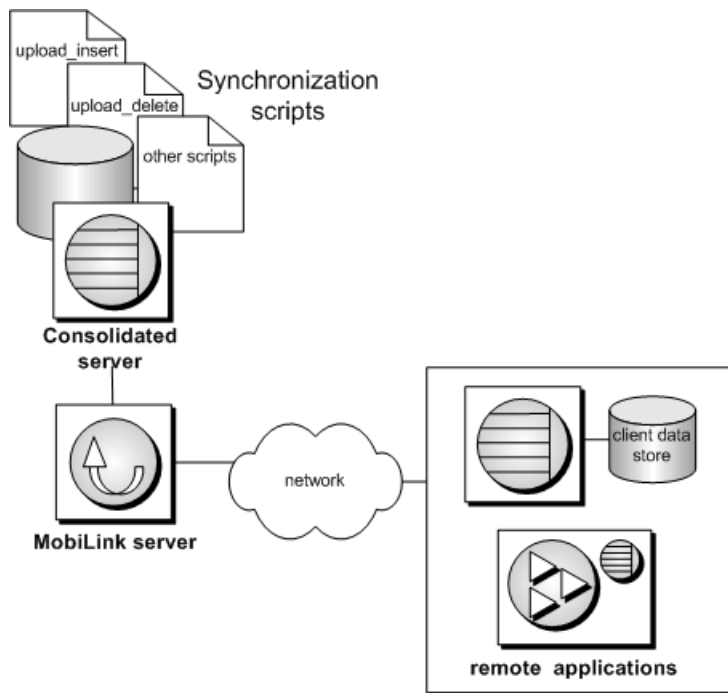
Scripts can be individual statements or stored procedure calls. They are stored or referenced in your consolidated database. To add scripts to the consolidated database, you can use SQL Central or you can use system procedures.

#### 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.

#### In this section:

##### [Simple Synchronization Script Example \[page 290\]](#)

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.

##### [Scripts and the Synchronization Process \[page 291\]](#)

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.

##### [Script Types \[page 292\]](#)

There are two types of synchronization scripts, connection-level scripts and table-level scripts.

##### [Script Parameters \[page 294\]](#)

Most synchronization scripts can receive parameters from the MobiLink server. For details about the parameters you can use in each script, see the documentation for synchronization events.

#### [Script Versions \[page 312\]](#)

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.

#### [Scripts Required for Synchronization \[page 315\]](#)

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.

#### [Script Additions and Deletions \[page 316\]](#)

When you use the *Create Synchronization Model Wizard*, scripts are automatically added to the consolidated database when you deploy the model.

#### [Scripts to Upload Rows \[page 321\]](#)

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.

#### [Scripts to Download Rows \[page 324\]](#)

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.

#### [Scripts to Handle Errors \[page 329\]](#)

An error in a synchronization script occurs when an operation in the script fails while the MobiLink server is executing it.

## Related Information

[The Synchronization Process](#)

[Synchronization Events \[page 332\]](#)

[Options for Writing Server-side Synchronization Logic](#)

[Synchronization Scripts in Microsoft .NET \[page 541\]](#)

[Synchronization Script Writing in Java \[page 526\]](#)

[Synchronization Techniques \[page 113\]](#)

### 1.12.1.1 Simple Synchronization Script Example

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. SQL 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.

## 1.12.1.2 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

An overview of MobiLink events is provided in the documentation for synchronization events.

## 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.

Reference material, including details about each script and its parameters, can be found in the documentation for synchronization events.

## Related Information

[Scripts to Upload Rows \[page 321\]](#)

[Scripts to Download Rows \[page 324\]](#)

[Synchronization Events \[page 332\]](#)

### 1.12.1.3 Script Types

There are two types of synchronization scripts, connection-level scripts and table-level 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.

#### **In this section:**

[Connection Scripts \[page 293\]](#)

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.

[Table Scripts \[page 293\]](#)

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.

### 1.12.1.3.1 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.

### Related Information

[Script Versions \[page 312\]](#)

### 1.12.1.3.2 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.

## 1.12.1.4 Script Parameters

Most synchronization scripts can receive parameters from the MobiLink server. For details about the parameters you can use in each script, see the documentation for synchronization events.

You can specify parameters in your SQL scripts in one of two ways:

- named script parameters
- question marks (deprecated in SQL scripts)

### In this section:

#### [Named Script Parameters \[page 294\]](#)

Named parameters have the following advantages over (deprecated) question marks:

#### [Script Parameters Represented by Question Marks \(Deprecated for SQL\) \[page 296\]](#)

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.

#### [Commenting Script Parameters \[page 296\]](#)

The following forms of comments are recognized:

#### [MobiLink System Parameters and Events \[page 297\]](#)

The following table provides a list of MobiLink system parameters that shows which event type each parameter is valid for and which event(s) each parameter can be used in.

#### [User-defined Named Parameters \[page 310\]](#)

You can also define your own parameters. These are especially useful for RDBMSs that don't allow user-defined variables.

#### [Authentication Parameters \[page 311\]](#)

In MobiLink scripts, authentication parameters can be specified using named parameters.

## Related Information

[Synchronization Events \[page 332\]](#)

### 1.12.1.4.1 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, called user-defined named parameters.

You cannot mix named parameters and question marks in a single script.

There are five types of MobiLink named parameters. To specify a named parameter, you must prefix it with its type, as follows:

Type of named parameter	Prefix	Examples
System parameter.	s.	{ml s.remote_id}
Row parameter. Describes the column name. If the column name contains spaces, enclose it in double quotes or square brackets.	r.	{ml r.cust_id} {ml r."Column name"}
Old row parameter. Only used in upload_update scripts to specify the pre-image column values. If the column name contains spaces, enclose it in double quotes or square brackets.	o.	{ml o.cust_name} {ml o."Column name"}
Authentication parameter.	a.	{ml a.1}
User-defined parameter. If the parameter will be updated (in/out), use the prefix u. If the parameter will only be referenced (input-only), use the prefix ui.	u. or ui.	{ml u.varname} {ml ui.varname}

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 ( id, c0 ) VALUES({ml r.id, r.c0})
```

and

```
{ml INSERT INTO t ( id, c0 ) VALUES( r.id, r.c0 ) }
```

## Related Information

[Authentication Parameters \[page 311\]](#)

[User-defined Named Parameters \[page 310\]](#)

## 1.12.1.4.2 Script Parameters Represented by Question Marks (Deprecated for SQL)

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.

### i Note

Representing parameters with question marks has been deprecated in SQL scripts. It is strongly recommended that you use named parameters instead.

## Related Information

[Named Script Parameters \[page 294\]](#)

[User-defined Named Parameters \[page 310\]](#)

[Synchronization Events \[page 332\]](#)

## 1.12.1.4.3 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.



## 1.12.1.4.4 MobiLink System Parameters and Events

The following table provides a list of MobiLink system parameters that shows which event type each parameter is valid for and which event(s) each parameter can be used in.

System parameter	Event type	Event parameter is available in
action_code	connection	The action_code parameter is available in the following events: <ul style="list-style-type: none"><li>• handle_error</li><li>• handle_odbc_error</li><li>• report_error</li><li>• report_odbc_error</li></ul>
authentication_message	connection	The authentication_message parameter is available in the following events: <ul style="list-style-type: none"><li>• authenticate_parameters</li><li>• authenticate_user</li><li>• authenticate_user_hashed</li></ul>
authentication_status	connection	The authentication_status parameter is available in the following events: <ul style="list-style-type: none"><li>• authenticate_parameters</li><li>• authenticate_user</li><li>• authenticate_user_hashed</li></ul>
bytes	connection and table	The bytes parameter is available in the following events: <ul style="list-style-type: none"><li>• download_statistics</li><li>• upload_statistics</li></ul>
conflicted_updates	connection and table	The conflicted_updates parameter is available in the following events: <ul style="list-style-type: none"><li>• upload_statistics</li></ul>
connection_retries	connection	The conflicted_retries parameter is available in the following events: <ul style="list-style-type: none"><li>• synchronization_statistics</li></ul>
deadlocks	connection and table	The deadlocks parameter is available in the following events: <ul style="list-style-type: none"><li>• synchronization_statistics (connection event)</li><li>• upload_statistics (connection and table events)</li></ul>

System parameter	Event type	Event parameter is available in
deleted_rows	connection and table	The deleted_rows parameter is available in the following events: <ul style="list-style-type: none"> <li>download_statistics</li> <li>upload_statistics</li> </ul>
error_code	connection	The error_code parameter is available in the following events: <ul style="list-style-type: none"> <li>handle_error</li> <li>modify_error_message</li> <li>report_error</li> </ul>
error_message	connection	The error_message parameter is available in the following events: <ul style="list-style-type: none"> <li>handle_error</li> <li>handle_odbc_error</li> <li>modify_error_message</li> <li>report_error</li> <li>report_odbc_error</li> </ul>
errors	connection and table	The errors parameter is available in the following events: <ul style="list-style-type: none"> <li>download_statistics</li> <li>synchronization_statistics</li> <li>upload_statistics</li> </ul>
event_name	connection and table	The event_name parameter is available in the following events: <ul style="list-style-type: none"> <li>time_statistics</li> </ul>
fetches_rows	connection and table	The fetched_rows parameter is available in the following events: <ul style="list-style-type: none"> <li>download_statistics</li> </ul>
file_authentication_code	connection	The file_authentication_code parameter is available in the following events: <ul style="list-style-type: none"> <li>authenticate_file_transfer</li> <li>authenticate_file_upload</li> </ul>
filename	connection	The filename parameter is available in the following events: <ul style="list-style-type: none"> <li>authenticate_file_transfer</li> <li>authenticate_file_upload</li> </ul>

System parameter	Event type	Event parameter is available in
file_size	connection	The file_size parameter is available in the following events: <ul style="list-style-type: none"> <li>authenticate_file_upload</li> </ul>
filtered_rows	connection and table	The filtered_rows parameter is available in the following events: <ul style="list-style-type: none"> <li>download_statistics</li> </ul>
generation_number	connection	The filtered_rows parameter is available in the following events: <ul style="list-style-type: none"> <li>begin_publication</li> <li>end_publication</li> </ul>
hashed_new_password	connection	The hashed_new_password parameter is available in the following events: <ul style="list-style-type: none"> <li>authenticate_user_hashed</li> </ul>
hashed_password	connection	The hashed_password parameter is available in the following events: <ul style="list-style-type: none"> <li>authenticate_user_hashed</li> </ul>
ignored_deletes	connection and table	The ignored_deletes parameter is available in the following events: <ul style="list-style-type: none"> <li>upload_statistics</li> </ul>
ignored_inserts	connection and table	The ignored_inserts parameter is available in the following events: <ul style="list-style-type: none"> <li>upload_statistics</li> </ul>
ignored_updates	connection and table	The ignored_updates parameter is available in the following events: <ul style="list-style-type: none"> <li>upload_statistics</li> </ul>
inserted_rows	connection and table	The inserted_rows parameter is available in the following events: <ul style="list-style-type: none"> <li>upload_statistics</li> </ul>

System parameter	Event type	Event parameter is available in
last_download	connection	<p>The last_download parameter is available in the following events:</p> <ul style="list-style-type: none"> <li>• begin_download</li> <li>• end_download</li> <li>• modify_last_download_timestamp</li> <li>• modify_next_last_download_timestamp</li> <li>• nonblocking_download_ack</li> <li>• prepare_for_download</li> </ul>
last_publication_download	connection	<p>The last_publication_download parameter is available in the following events:</p> <ul style="list-style-type: none"> <li>• begin_publication</li> <li>• end_publication</li> <li>• publication_nonblocking_download_ack</li> </ul>
last_publication_upload	connection	<p>The last_publication_upload parameter is available in the following events:</p> <ul style="list-style-type: none"> <li>• begin_publication</li> <li>• end_publication</li> </ul>
last_table_download	table	<p>The last_table_download parameter is available in the following events:</p> <ul style="list-style-type: none"> <li>• begin_download</li> <li>• begin_download_deletes</li> <li>• begin_download_rows</li> <li>• download_cursor</li> <li>• download_delete_cursor</li> <li>• end_download</li> <li>• end_download_deletes</li> <li>• end_download_rows</li> </ul>
maximum_time	connection and table	<p>The maximum_time parameter is available in the following events:</p> <ul style="list-style-type: none"> <li>• time_statistics</li> </ul>
minimum_time	connection and table	<p>The minimum_time parameter is available in the following events:</p> <ul style="list-style-type: none"> <li>• time_statistics</li> </ul>
new_password	connection	<p>The new_password parameter is available in the following events:</p> <ul style="list-style-type: none"> <li>• authenticate_user</li> </ul>

System parameter	Event type	Event parameter is available in
new_remote_id	connection and table	The new_remote_id parameter is available in the following events: <ul style="list-style-type: none"> <li>authenticate_user</li> <li>begin_synchronization</li> </ul>
new_username	connection and table	The new_username parameter is available in the following events: <ul style="list-style-type: none"> <li>authenticate_user</li> <li>begin_synchronization</li> </ul>
next_last_download	connection	The next_last_download parameter is available in the following events: <ul style="list-style-type: none"> <li>generate_next_last_download_timestamp</li> <li>modify_next_last_download_timestamp</li> </ul>
number_of_calls	connection and table	The number_of_calls parameter is available in the following events: <ul style="list-style-type: none"> <li>time_statistics</li> </ul>
odbc_state	connection	The odbc_state parameter is available in the following events: <ul style="list-style-type: none"> <li>handle_odbc_error</li> <li>report_odbc_error</li> </ul>
password	connection	The password parameter is available in the following events: <ul style="list-style-type: none"> <li>authenticate_user</li> </ul>
publication_name	connection	The publication_name parameter is available in the following events: <ul style="list-style-type: none"> <li>begin_publication</li> <li>end_publication</li> <li>publication_nonblocking_download_ack</li> </ul>

System parameter	Event type	Event parameter is available in
remote_id	connection and table	<p>The remote_id parameter is available in the following events:</p> <ul style="list-style-type: none"> <li>• authenticate_parameters (connection event)</li> <li>• authenticate_user (connection event)</li> <li>• authenticate_user_hashed (connection event)</li> <li>• begin_download (connection and table event)</li> <li>• begin_download_deletes (table event)</li> <li>• begin_download_rows (table event)</li> <li>• begin_publication (connection event)</li> <li>• begin_synchronization (connection and table event)</li> <li>• begin_upload (connection and table event)</li> <li>• begin_upload_deletes (table event)</li> <li>• begin_upload_rows (table event)</li> <li>• download_cursor (table event)</li> <li>• download_delete_cursor (table event)</li> <li>• download_statistics (connection and table event)</li> <li>• end_download (connection and table event)</li> <li>• end_download_deletes (table event)</li> <li>• end_download_rows (table event)</li> <li>• end_publication (connection event)</li> <li>• end_synchronization (connection and table event)</li> <li>• end_upload (connection and table event)</li> <li>• end_upload_deletes (table event)</li> <li>• end_upload_rows (table event)</li> <li>• generate_next_last_download_timestamp (connection)</li> <li>• handle_error (connection event)</li> <li>• handle_odbc_error (connection event)</li> <li>• modify_error_message (connection event)</li> </ul>

System parameter	Event type	Event parameter is available in
		<ul style="list-style-type: none"> <li>• modify_last_download_timestamp (connection event)</li> <li>• modify_next_last_download_timestamp (connection event)</li> <li>• modify_user (connection event)</li> <li>• nonblocking_download_ack (connection event)</li> <li>• prepare_for_download (connection event)</li> <li>• publication_nonblocking_download_ack (connection event)</li> <li>• report_error (connection event)</li> <li>• report_odbc_error (connection event)</li> <li>• resolve_conflict (table event)</li> <li>• synchronization_statistics (connection and table event)</li> <li>• time_statistics (connection and table event)</li> <li>• upload_delete (table event)</li> <li>• upload_fetch (table event)</li> <li>• upload_fetch_column_conflict (table event)</li> <li>• upload_insert (table event)</li> <li>• upload_new_row_insert (table event)</li> <li>• upload_old_row_insert (table event)</li> <li>• upload_statistics (connection and table event)</li> <li>• upload_update (table event)</li> </ul>
remote_key	connection	<p>The remote_key parameter is available in the following events:</p> <ul style="list-style-type: none"> <li>• authenticate_file_transfer</li> <li>• authenticate_file_upload</li> </ul>

System parameter	Event type	Event parameter is available in
script_version	connection and table	<p>The script_version parameter is available in the following events:</p> <ul style="list-style-type: none"> <li>• authenticate_file_transfer (connection event)</li> <li>• authenticate_file_upload (connection event)</li> <li>• authenticate_parameters (connection event)</li> <li>• authenticate_user (connection event)</li> <li>• authenticate_user_hashed (connection event)</li> <li>• begin_download (connection and table event)</li> <li>• begin_download_deletes (table event)</li> <li>• begin_download_rows (table event)</li> <li>• begin_publication (connection event)</li> <li>• begin_synchronization (connection and table event)</li> <li>• begin_upload (connection and table event)</li> <li>• begin_upload_deletes (table event)</li> <li>• begin_upload_rows (table event)</li> <li>• download_cursor (table event)</li> <li>• download_delete_cursor (table event)</li> <li>• download_statistics (connection and table event)</li> <li>• end_download (connection and table event)</li> <li>• end_download_deletes (table event)</li> <li>• end_download_rows (table event)</li> <li>• end_publication (connection event)</li> <li>• end_synchronization (connection and table event)</li> <li>• end_upload (connection and table event)</li> <li>• end_upload_deletes (table event)</li> <li>• end_upload_rows (table event)</li> <li>• generate_next_last_download_timestamp (connection)</li> </ul>



System parameter	Event type	Event parameter is available in
		<ul style="list-style-type: none"> <li>• handle_DownloadData (connection event)</li> <li>• handle_error (connection event)</li> <li>• handle_odbc_error (connection event)</li> <li>• modify_error_message (connection event)</li> <li>• modify_last_download_timestamp (connection event)</li> <li>• modify_next_last_download_timestamp (connection event)</li> <li>• modify_user (connection event)</li> <li>• nonblocking_download_ack (connection event)</li> <li>• prepare_for_download (connection event)</li> <li>• publication_nonblocking_download_ack (connection event)</li> <li>• report_error (connection event)</li> <li>• report_odbc_error (connection event)</li> <li>• resolve_conflict (table event)</li> <li>• synchronization_statistics (connection and table event)</li> <li>• time_statistics (connection and table event)</li> <li>• upload_delete (table event)</li> <li>• upload_fetch (table event)</li> <li>• upload_fetch_column_conflict (table event)</li> <li>• upload_insert (table event)</li> <li>• upload_new_row_insert (table event)</li> <li>• upload_old_row_insert (table event)</li> <li>• upload_statistics (connection and table event)</li> <li>• upload_update (table event)</li> </ul>

**i Note**  
 The script version can be accessed from Java and .NET scripts using the getVersion method on the DBConnectionContext classes.

<b>System parameter</b>	<b>Event type</b>	<b>Event parameter is available in</b>
subdir	connection	The subdir parameter is available in the following events: <ul style="list-style-type: none"> <li>• authenticate_file_transfer</li> <li>• authenticate_file_upload</li> </ul>
subscription_id	connection	The subscription_id parameter is available in the following events: <ul style="list-style-type: none"> <li>• begin_publication</li> <li>• end_publication</li> <li>• publication_nonblocking_download_ack</li> </ul>
synchronization_ok	connection and table	The synchronization_ok parameter is available in the following events: <ul style="list-style-type: none"> <li>• end_synchronization</li> </ul>
synchronized_tables	connection	The synchronized_tables parameter is available in the following events: <ul style="list-style-type: none"> <li>• synchronization_statistics</li> </ul>

System parameter	Event type	Event parameter is available in
table	connection and table	<p>The table parameter is available in the following events:</p> <ul style="list-style-type: none"> <li>• begin_download (table event)</li> <li>• begin_download_deletes (table event)</li> <li>• begin_download_rows (table event)</li> <li>• begin_synchronization (table event)</li> <li>• begin_upload (table event)</li> <li>• begin_upload_deletes (table event)</li> <li>• begin_upload_rows (table event)</li> <li>• download_statistics (table event)</li> <li>• end_download (table event)</li> <li>• end_download_deletes (table event)</li> <li>• end_download_rows (table event)</li> <li>• end_synchronization (table event)</li> <li>• end_upload (table event)</li> <li>• end_upload_deletes (table event)</li> <li>• end_upload_rows (table event)</li> <li>• handle_error (connection event)</li> <li>• handle_odbc_error (connection event)</li> <li>• report_error (connection event)</li> <li>• report_odbc_error (connection event)</li> <li>• resolve_conflict (table event)</li> <li>• synchronization_statistics (table event)</li> <li>• time_statistics (table event)</li> <li>• upload_statistics (table event)</li> </ul>
total_time	connection and table	<p>The total_time parameter is available in the following events:</p> <ul style="list-style-type: none"> <li>• time_statistics</li> </ul>
updated_rows	connection and table	<p>The updated_rows parameter is available in the following events:</p> <ul style="list-style-type: none"> <li>• upload_statistics</li> </ul>

System parameter	Event type	Event parameter is available in
username	connection and table	<p>The username parameter is available in the following events:</p> <ul style="list-style-type: none"> <li>• authenticate_file_transfer (connection event)</li> <li>• authenticate_file_upload (connection event)</li> <li>• authenticate_parameters (connection event)</li> <li>• authenticate_user (connection event)</li> <li>• authenticate_user_hashed (connection event)</li> <li>• begin_download (connection and table event)</li> <li>• begin_download_deletes (table event)</li> <li>• begin_download_rows (table event)</li> <li>• begin_publication (connection event)</li> <li>• begin_synchronization (connection and table event)</li> <li>• begin_upload (connection and table event)</li> <li>• begin_upload_deletes (table event)</li> <li>• begin_upload_rows (table event)</li> <li>• download_cursor (table event)</li> <li>• download_delete_cursor (table event)</li> <li>• download_statistics (connection and table event)</li> <li>• end_download (connection and table event)</li> <li>• end_download_deletes (table event)</li> <li>• end_download_rows (table event)</li> <li>• end_publication (connection event)</li> <li>• end_synchronization (connection and table event)</li> <li>• end_upload (connection and table event)</li> <li>• end_upload_deletes (table event)</li> <li>• end_upload_rows (table event)</li> <li>• generate_next_last_download_timestamp (connection event)</li> </ul>

System parameter	Event type	Event parameter is available in
		<ul style="list-style-type: none"> <li>• handle_error (connection event)</li> <li>• handle_odbc_error (connection error)</li> <li>• modify_error_message (connection error)</li> <li>• modify_last_download_timestamp (connection error)</li> <li>• modify_next_last_download_timestamp (connection event)</li> <li>• modify_user (connection event)</li> <li>• nonblocking_download_ack (connection event)</li> <li>• prepare_for_download (connection event)</li> <li>• publication_nonblocking_download_ack (connection event)</li> <li>• report_error (connection event)</li> <li>• report_odbc_error (connection event)</li> <li>• resolve_conflict (table event)</li> <li>• synchronization_statistics (connection and table event)</li> <li>• time_statistics (connection and table event)</li> <li>• upload_delete (table event)</li> <li>• upload_fetch (table event)</li> <li>• upload_fetch_column_conflict (table event)</li> <li>• upload_insert (table event)</li> <li>• upload_new_row_insert (table event)</li> <li>• upload_old_row_insert (table event)</li> <li>• upload_statistics (connection and table event)</li> <li>• upload_update (table event)</li> </ul>
warnings	connection and table	<p>The warnings parameter is available in the following events:</p> <ul style="list-style-type: none"> <li>• download_statistics</li> <li>• synchronization_statistics</li> <li>• upload_statistics</li> </ul>

## 1.12.1.4.5 User-defined Named Parameters

You can also define your own parameters. These are especially useful for RDBMSs that don't allow user-defined variables.

User-defined parameters are defined (and set to null) when first referenced. They must start with ui and a period (ui.) if the parameter will only be referenced (input-only) and u and a period (u.) if the parameter will be updated (in/out). A user-defined parameter lasts for one synchronization; it is set to null at the start of every synchronization.

A typical use of user-defined parameters is to access state information without having to store it in a table (requiring potentially complex joins).

### Example

For example, assume you create a stored procedure called MyCustomProc that sets a variable called var1 to 'custom\_value':

```
CREATE PROCEDURE MyCustomProc(
  IN username VARCHAR(128), INOUT var1 VARCHAR(128)
)
begin
  SET var1 = 'custom_value';
end
```

The following begin\_synchronization script defines the user-defined parameter var1 and sets the value to 'custom\_value':

```
CALL ml_add_connection_script (
  'version1',
  'begin_synchronization',
  '{call MyCustomProc( {ml s.username}, {ml u.var1} )}' );
```

The following download\_cursor table script references var1, whose value is 'custom\_value':

```
CALL ml_add_table_script (
  'version1',
  'MyTable',
  'download_cursor',
  'select pk, coll from MyTable where u_name = {ml s.username} and
  some_other_column = {ml ui.var1}' );
```

Assume you have another stored procedure called MyPFDFProc that defines its first parameter as INOUT. The following prepare\_for\_download script changes the value of var1 to 'pfd\_value':

```
CALL ml_add_connection_script (
  'version1',
  'prepare_for_download',
  '{call MyPFDFProc( {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} )' );
```

### 1.12.1.4.6 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.

#### Example

For UltraLite remote databases, pass the parameters using the num\_auth\_parms and auth\_parms fields in the ul\_sync\_info struct. num\_auth\_parms is a count of the number of parameters, from 0 to 255. auth\_parms is a pointer to an array of strings. During synchronization the authentication parameters are obfuscated in the same way as passwords. If num\_auth\_parms is 0, set auth\_parms to null. The following is an example of passing parameters in UltraLite:

```
ul_char * Params[ 3 ] = { UL_TEXT( "param1" ),
    UL_TEXT( "param2" ), UL_TEXT( "param3" ) };
...
info.num_auth_parms = 3;
info.auth_parms = Params;
```

For SQL Anywhere remote databases, you pass authentication parameters using the dbmlsync -ap option, in a comma-separated list. For example, the following command line passes three parameters:

```
dbmlsync -ap "param1,param2,param3"
```

On the server, you reference the authentication parameters using the order in which they were sent up. In this example, the authenticate\_parameters script could be:

```
CALL my_auth_parm (
    {ml s.authentication_status},
    {ml s.remote_id},
    {ml s.username},
    {ml a.1},
    {ml a.2},
    {ml a.3}
)
```

## Related Information

[-ap dbmsync Option](#)

[Authentication Parameters Synchronization Parameter](#)

[Number of Authentication Parameters Synchronization Parameter](#)

### 1.12.1.5 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.

#### 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

Use the `ml_add_property` system procedure to add or delete MobiLink properties for your script version that can be referenced from classes in .NET or Java synchronization logic.

#### 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 SQL Central, you are prompted for the script version name.

You cannot use the following names for script versions: *ml\_sis\_1* or *ml\_qa\_1*. These names are used internally by MobiLink.

#### ⚠ Caution

It is strongly recommended that your script version names do not start with *ml\_*. Script versions starting with *ml\_* are reserved for internal use.

## Specifying a Script Version for a Synchronization

If no script version is specified at the remote site when synchronization is initiated, the synchronization fails.

### 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.

#### In this section:

##### [Adding a Script Version to a Consolidated Database \[page 314\]](#)

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

##### [Removing a Script Version From a Consolidated Database \[page 315\]](#)

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

## Related Information

[Script Versions and Subscriptions](#)

[ml\\_add\\_property System Procedure \[page 605\]](#)

## 1.12.1.5.1 Adding a Script Version to a Consolidated Database

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

### Context

When adding scripts with system procedures, if you specify a new script version name it is automatically added with the script.

In SQL Central, only one script version is allowed per synchronization model and it is by default given the same name as the synchronization model.

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.

### Procedure

1. From the *View* menu, click *Folders*.
2. In the left pane of SQL 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.

### Related Information

[MobiLink Server System Procedures \[page 582\]](#)

## 1.12.1.5.2 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.

### Procedure

1. From the *View* menu, click *Folders*.
2. In the left pane of SQL 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*.

### Results

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

## 1.12.1.6 Scripts Required for Synchronization

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}`.

## Related Information

[Ignored Scripts \[page 320\]](#)

### 1.12.1.7 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 SQL 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.

If you are using SQL Central, you must add a script version to the database before you can add individual scripts.

### 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.

The following stored procedures can be used to add or delete scripts:

- 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

#### In this section:

[Adding a Connection Script \[page 317\]](#)

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

[Deleting a Connection Script \[page 318\]](#)

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

[Adding a Table Script \[page 319\]](#)

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

[Deleting a Table Script \[page 319\]](#)

Use the following procedure to delete a table script.

[Direct Inserts of Scripts \[page 320\]](#)

Use stored procedures or SQL Central to insert scripts into the system tables.

[Ignored Scripts \[page 320\]](#)

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 issues an error and aborts the synchronization.

## Related Information

[MobiLink Server System Tables \[page 156\]](#)

[Adding a Script Version to a Consolidated Database \[page 314\]](#)

[ml\\_add\\_connection\\_script System Procedure \[page 589\]](#)

[ml\\_add\\_table\\_script System Procedure \[page 609\]](#)

[ml\\_add\\_dnet\\_connection\\_script System Procedure \[page 591\]](#)

[ml\\_add\\_dnet\\_table\\_script System Procedure \[page 592\]](#)

[ml\\_add\\_java\\_connection\\_script System Procedure \[page 593\]](#)

[ml\\_add\\_java\\_table\\_script System Procedure \[page 595\]](#)



### 1.12.1.7.1 Adding a Connection Script

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

#### Prerequisites

If you are using SQL Central, you must add a script version to the database before you can add individual scripts.

#### Procedure

1. From the *View* menu, click *Folders*.
2. In the left pane of SQL 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* .
4. Follow the instructions in the *Create Connection Script Wizard*.

## Results

The connection script is created.

## Related Information

[Adding a Script Version to a Consolidated Database \[page 314\]](#)

### 1.12.1.7.2 Deleting a Connection Script

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

#### Procedure

1. From the *View* menu, click *Folders*.
2. In the left pane of SQL 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.

### 1.12.1.7.3 Adding a Table Script

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

#### Procedure

1. From the *View* menu, click *Folders*.
2. In the left pane of SQL 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.

### 1.12.1.7.4 Deleting a Table Script

Use the following procedure to delete a table script.

#### Procedure

1. From the *View* menu, click *Folders*.
2. In the left pane of SQL 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.

### 1.12.1.7.5 Direct Inserts of Scripts

Use stored procedures or SQL 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.

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 %SQLANY17%\MobiLink\Setup and are called:

Consolidated database	Setup file
Adaptive Server Enterprise	syncase.sql
IBM DB2 LUW (deprecated)	syncdb2.sql
Microsoft Azure	syncmss.sql
Microsoft SQL Server	syncmss.sql
MySQL	syncmys.sql
Oracle	syncora.sql
SAP HANA	synchana.sql
SAP IQ	synciq.sql
SQL Anywhere	syncsa.sql

## Related Information

[MobiLink Server System Tables \[page 156\]](#)

### 1.12.1.7.6 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 issues an error 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.



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.

### 1.12.1.8 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.

#### 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.

#### In this section:

##### [upload\\_insert Scripts \[page 322\]](#)

The MobiLink server uses this event during processing of the upload to handle rows inserted into the remote database.

##### [upload\\_update Scripts \[page 322\]](#)

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.

##### [upload\\_delete Scripts \[page 323\]](#)

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.

##### [upload\\_fetch Scripts \[page 324\]](#)

The `upload_fetch` script is a SELECT statement that defines a cursor in the consolidated database table.

## Related Information

[Events During Upload \[page 342\]](#)

[.NET Synchronization Techniques \[page 553\]](#)

[Transaction Log Files](#)

### 1.12.1.8.1 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.

## Related Information

[upload\\_insert Table Event \[page 506\]](#)

### 1.12.1.8.2 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.

## Related Information

[upload\\_update Table Event \[page 522\]](#)

### 1.12.1.8.3 upload\_delete Scripts

The MobiLink server uses this event during processing of the upload to handle rows deleted from the remote database. The following statement shows how to use the upload\_delete statement.

```
DELETE FROM emp
WHERE emp_id = {ml r.emp_id};
```

## Notes

- When using question marks instead of named parameters as placeholders, the number of parameters must be equal to one of the following (the use of question marks in SQL scripts has been deprecated):
  - The number of primary key columns.
  - The number of all columns.

## Related Information

[upload\\_delete Table Event \[page 500\]](#)

## 1.12.1.8.4 upload\_fetch Scripts

The `upload_fetch` script is a `SELECT` statement that defines a cursor in the consolidated database table.

This cursor is used to compare the old values of updated rows, as received from the remote database, against the current value in the consolidated database. In this way, the `upload_fetch` script identifies conflicts when updates are being processed.

Given a synchronized table defined as:

```
CREATE TABLE uf_example (
  pk1 integer NOT NULL,
  pk2 integer NOT NULL,
  val varchar(200),
  PRIMARY KEY( pk1, pk2 ));
```

Then one possible `upload_fetch` script for this table is:

```
SELECT pk1, pk2, val
FROM uf_example
WHERE pk1 = {m1 r.pk1} AND pk2 = {m1 r.pk2}
```

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.

## Related Information

[upload\\_fetch Table Event \[page 502\]](#)

## 1.12.1.9 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.

The MobiLink server downloads data in a single transaction.

## 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}
```

- At the end of the download processing, the client automatically deletes rows if necessary to avoid referential integrity violations.
- If you change the `SendDownloadAck` setting to `ON`, the download transaction is committed but the acknowledgement scripts are not executed until the acknowledgement is received. By default, `SendDownloadAck` is set to `OFF`.

### ⚠ 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.

#### In this section:

##### [download\\_cursor Scripts \[page 326\]](#)

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.

##### [download\\_delete\\_cursor Scripts \[page 327\]](#)

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}`.

## Related Information

[Result Sets from Stored Procedure Calls \[page 147\]](#)

[Events During Download \[page 345\]](#)

[Last Download Times in Scripts \[page 117\]](#)

[Referential Integrity and Synchronization](#)

[SendDownloadAck \(sa\) Extended Option](#)

[Send Download Acknowledgement Synchronization Parameter](#)

[nonblocking\\_download\\_ack Connection Event \[page 466\]](#)

[publication\\_nonblocking\\_download\\_ack Connection Event \[page 472\]](#)

## 1.12.1.9.1 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_dnl_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.

### Example

The following script could serve as a `download_cursor` script for a remote table that holds employee information. This script downloads information about all the employees.

```
SELECT emp_id, emp_fname, emp_lname
FROM employee;
```

The MobiLink server passes specific parameters to some scripts. The MobiLink server substitutes the value of the parameter before executing the statement 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:

```
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 SQL Central to view the remote tables listed together with their scripts.

## Related Information

[Partitioned Rows Among Remote Databases \[page 121\]](#)

[download\\_delete\\_cursor Scripts \[page 327\]](#)

[download\\_cursor Table Event \[page 393\]](#)

[ml\\_add\\_missing\\_dnl\\_d\\_scripts System Procedure \[page 600\]](#)

### 1.12.1.9.2 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\_dnl\_d\_scripts system procedure to define missing download scripts as ignored.

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.

## 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, Microsoft Azure, and Microsoft SQL Server databases.

```
SELECT NULL, NULL
```

In IBM DB2 LUW and Oracle consolidated databases, specify a dummy table to select null. For IBM DB2 LUW 9.5 and 9.7, you can use the following syntax:

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

### Note

Support for IBM DB2 consolidated databases is deprecated.

For Oracle consolidated databases, use the following syntax:

```
SELECT NULL, NULL FROM DUAL
```

For SAP HANA consolidated databases, use the following syntax:

```
SELECT NULL, NULL, FROM DUMMY
```

## Example

The following example is a download\_delete\_cursor script for a remote table that holds employee information. The MobiLink server uses this SQL statement to define the delete cursor. This script deletes information about all orders that are both in the consolidated and remote databases at the time the script is executed.

```
SELECT order_id
FROM UOrder
```



The `download_delete_cursor` accepts the parameters `last_table_download` and `username`. The following script shows how you can use each parameter to narrow your selection.

```
SELECT order_id
FROM ULOrder
WHERE last_modified >= {ml s.last_table_download}
   AND status = 'Approved'
   AND user_name = {ml s.username}
```

Another strategy is to allow the client application to delete the rows itself. This method is possible only when a rule identifies the unnecessary rows. For example, rows might contain a timestamp that indicates an expiry date. Before you delete the rows at the remote, use the `STOP SYNCHRONIZATION DELETE` statement to stop these deletes being uploaded during the next synchronization. Be sure to execute `START SYNCHRONIZATION DELETE` immediately afterward if you want other deletes to be synchronized in the normal fashion.

## Notes

- The `download_delete_cursor` script must contain primary key columns in the same order as they are defined in the remote database.
- You can use the referential integrity checking built into all MobiLink clients to delete rows in an efficient manner by deleting only the parent rows.

## Related Information

[Deletes \[page 142\]](#)

[Partitioned Rows Among Remote Databases \[page 121\]](#)

[Snapshot Synchronization \[page 119\]](#)

[Referential Integrity and Synchronization](#)

[Temporarily Stopping the Synchronization of Deletes \[page 143\]](#)

[ml\\_add\\_missing\\_dnl\\_d\\_scripts System Procedure \[page 600\]](#)

[download\\_cursor Table Event \[page 393\]](#)

[download\\_delete\\_cursor Table Event \[page 396\]](#)

[STOP SYNCHRONIZATION DELETE Statement \[MobiLink\]](#)

### 1.12.1.10 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_odbc_error` script.

If the `handle_error` script itself causes a SQL error, then the default action code (3000) is assumed.

### In this section:

[Error Reporting \[page 331\]](#)

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.

## Related Information

[handle\\_error Connection Event \[page 440\]](#)

[handle\\_odbc\\_error Connection Event \[page 445\]](#)

[report\\_error Connection Event \[page 476\]](#)

[report\\_odbc\\_error Connection Event \[page 480\]](#)

## 1.12.1.10.1 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.

### Example

The following `report_error` script, which consists of a single insert statement, adds the script parameters into a table, along with the current date and time. The script does not commit this change because the MobiLink server always does so automatically.

```
INSERT INTO errors
VALUES (
  CURRENT DATE,
  {ml s.action_code},
  {ml s.error_code},
  {ml s.error_message},
  {ml s.username},
  {ml s.table} );
```

### Related Information

[handle\\_error Connection Event \[page 440\]](#)

[handle\\_odbc\\_error Connection Event \[page 445\]](#)

[report\\_error Connection Event \[page 476\]](#)

[report\\_odbc\\_error Connection Event \[page 480\]](#)

## 1.12.2 Synchronization Events

A MobiLink synchronization is made up of many events.

### In this section:

#### [Overview of MobiLink Events \[page 336\]](#)

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.

#### [Data Scripts \[page 346\]](#)

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.

#### [authenticate\\_file\\_transfer Connection Event \[page 348\]](#)

Implements custom authentication for file transfers using the `mlfiletransfer` utility or the `MLFileDownload` method.

#### [authenticate\\_file\\_upload Connection Event \[page 349\]](#)

Implements custom authentication for file transfers using the `mlfiletransfer` utility or the `MLFileUpload` method.

#### [authenticate\\_parameters Connection Event \[page 351\]](#)

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.

#### [authenticate\\_user Connection Event \[page 354\]](#)

Implements custom user authentication.

#### [authenticate\\_user\\_hashed Connection Event \[page 360\]](#)

Implements a custom user authentication mechanism.

#### [begin\\_connection Connection Event \[page 364\]](#)

Invoked at the time the MobiLink server connects to the consolidated database server.

#### [begin\\_connection\\_autocommit Connection Event \[page 365\]](#)

Invoked at the time MobiLink server connects to the consolidated database server, temporarily allowing you to execute a script when autocommit is on.

#### [begin\\_download Connection Event \[page 366\]](#)

Processes any statements just before the MobiLink server commences preparing the download.

#### [begin\\_download Table Event \[page 368\]](#)

Processes statements related to a specific table just before preparing the download inserts, updates, and deletions.

#### [begin\\_download\\_deletes Table Event \[page 371\]](#)

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.

#### [begin\\_download\\_rows Table Event \[page 372\]](#)

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.

#### [begin\\_publication Connection Event \[page 374\]](#)

Provides useful information about the publication(s) being synchronized. This script may also be used to manage generation numbers for file-based downloads.

[begin\\_synchronization Connection Event \[page 377\]](#)

Processes statements in preparation for the synchronization process.

[begin\\_synchronization Table Event \[page 380\]](#)

Processes statements related to a specific table at the beginning of the synchronization.

[begin\\_upload Connection Event \[page 383\]](#)

Processes any statements just before the MobiLink server commences processing the uploaded inserts, updates, and deletes.

[begin\\_upload Table Event \[page 385\]](#)

Processes statements related to a specific table just before the MobiLink server commences processing the uploaded inserts, updates, and deletes.

[begin\\_upload\\_deletes Table Event \[page 388\]](#)

Processes statements related to a specific table just before uploading deleted rows from the specified table in the remote database.

[begin\\_upload\\_rows Table Event \[page 391\]](#)

Processes statements related to a specific table just before uploading inserts and updates from the specified table in the remote database.

[download\\_cursor Table Event \[page 393\]](#)

A data script that defines a cursor to select rows to download and insert and update in the given table in the remote database.

[download\\_delete\\_cursor Table Event \[page 396\]](#)

A data script that defines a cursor to select rows that are to be deleted in the remote database.

[download\\_statistics Connection Event \[page 398\]](#)

Provides access to synchronization statistics for download operations.

[download\\_statistics Table Event \[page 401\]](#)

Provides access to synchronization statistics for download operations by table.

[end\\_connection Connection Event \[page 405\]](#)

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.

[end\\_download Connection Event \[page 407\]](#)

Processes any statements just after the MobiLink server concludes preparation of the download data.

[end\\_download Table Event \[page 409\]](#)

Processes statements related to a specific table just after the MobiLink server concludes preparing the download rows.

[end\\_download\\_deletes Table Event \[page 412\]](#)

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.

[end\\_download\\_rows Table Event \[page 413\]](#)

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.

[end\\_publication Connection Event \[page 415\]](#)

Provides useful information about the publication(s) being synchronized.

[end\\_synchronization Connection Event \[page 418\]](#)

Processes statements at the end of the synchronization process.

[end\\_synchronization Table Event \[page 421\]](#)

Processes statements at the end of the synchronization process.

[end\\_upload Connection Event \[page 424\]](#)

Processes any statements just after the MobiLink server concludes processing uploaded inserts, updates, and deletes.

[end\\_upload Table Event \[page 426\]](#)

Processes statements related to a specific table just after the MobiLink server concludes processing of uploaded inserts, updates, and deletions.

[end\\_upload\\_deletes Table Event \[page 429\]](#)

Processes statements related to a specific table just after applying deletes uploaded from the specified table in the remote database.

[end\\_upload\\_rows Table Event \[page 432\]](#)

Processes statements related to a specific table just after applying uploaded inserts and updates from the specified table in the remote database.

[generate\\_next\\_last\\_download\\_timestamp Connection Event \[page 435\]](#)

The script is used to invoke a user-defined algorithm to generate the next\_last\_download\_timestamp.

[handle\\_DownloadData Connection Event \[page 437\]](#)

A non-SQL data script used by direct row handling to create a set of rows to download.

[handle\\_error Connection Event \[page 440\]](#)

Executed whenever the MobiLink server encounters a SQL error while invoking a data script.

[handle\\_odbc\\_error Connection Event \[page 445\]](#)

Executed whenever the MobiLink server encounters an ODBC error while invoking a data script.

[handle\\_UploadData Connection Event \[page 448\]](#)

A non-SQL data script used by direct row handling to process uploaded rows.

[modify\\_error\\_message Connection Event \[page 454\]](#)

The script can be used to customize the message text (error, warning, and information) that is sent to remote databases.

[modify\\_last\\_download\\_timestamp Connection Event \[page 457\]](#)

The script can be used to modify the last download time for the current synchronization.

[modify\\_next\\_last\\_download\\_timestamp Connection Event \[page 460\]](#)

The script can be used to modify the last download time for the next synchronization.

[modify\\_user Connection Event \[page 463\]](#)

Modify the MobiLink user name.

[nonblocking\\_download\\_ack Connection Event \[page 466\]](#)

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.

[prepare\\_for\\_download Connection Event \[page 470\]](#)

Processes any required operations between the upload and download transactions.

[publication\\_nonblocking\\_download\\_ack Connection Event \[page 472\]](#)

When you use download acknowledgement, this script provides a place to record the information that a publication has been successfully downloaded.

[report\\_error Connection Event \[page 476\]](#)

Allows you to log errors and to record the actions selected by the handle\_error script.

[report\\_odbc\\_error Connection Event \[page 480\]](#)

Allows you to log errors and to record the actions selected by the handle\_odbc\_error script.

[resolve\\_conflict Table Event \[page 483\]](#)

Defines a process for resolving a conflict in a specific table.

[synchronization\\_statistics Connection Event \[page 487\]](#)

Tracks synchronization statistics.

[synchronization\\_statistics Table Event \[page 491\]](#)

Provides access to synchronization statistics.

[time\\_statistics Connection Event \[page 494\]](#)

Tracks time statistics by user and event.

[time\\_statistics Table Event \[page 497\]](#)

Tracks time statistics.

[upload\\_delete Table Event \[page 500\]](#)

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.

[upload\\_fetch Table Event \[page 502\]](#)

A data script that fetches rows from a synchronized table in the consolidated database for row-level conflict detection.

[upload\\_fetch\\_column\\_conflict Table Event \[page 504\]](#)

A data script that fetches rows from a synchronized table in the consolidated database for column-level conflict detection.

[upload\\_insert Table Event \[page 506\]](#)

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.

[upload\\_new\\_row\\_insert Table Event \[page 508\]](#)

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.

[upload\\_old\\_row\\_insert Table Event \[page 510\]](#)

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.

[upload\\_statistics Connection Event \[page 513\]](#)

Provides access to synchronization statistics for upload operations.

[upload\\_statistics Table Event \[page 517\]](#)

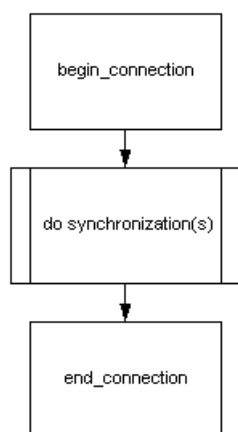
Provides access to synchronization statistics for upload operations for a specific table.

[upload\\_update Table Event \[page 522\]](#)

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.

## 1.12.2.1 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.

### In this section:

#### [Transactions Within a Synchronization \[page 337\]](#)

Within each synchronization, the following transactions may occur.

#### [The Upload Transaction \[page 337\]](#)

The upload transaction applies changes uploaded from a remote database.

#### [The Download Transaction \[page 338\]](#)

The download transaction fetches rows from the consolidated database. It begins with the `begin_download` event.

#### [The Non-Blocking Download Acknowledgement Transaction \[page 339\]](#)

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.

#### [MobiLink Event Model Notes \[page 339\]](#)

Keep the following points in mind:

#### [MobiLink Complete Event Model \[page 340\]](#)

This representation of the MobiLink event model assumes a full synchronization (not upload-only or download-only) with no errors.

#### [Events During Upload \[page 342\]](#)

The following pseudocode illustrates how upload events and upload scripts are invoked.



[Events During Download \[page 345\]](#)

The following pseudocode provides an overview of the sequence in which download events, and the script of the same name, are invoked.

### **1.12.2.1.1 Transactions Within a Synchronization**

Within each synchronization, the following transactions may occur.

- authentication
- begin synchronization
- upload
- prepare for download
- download
- end synchronization
- non-blocking download acknowledgement

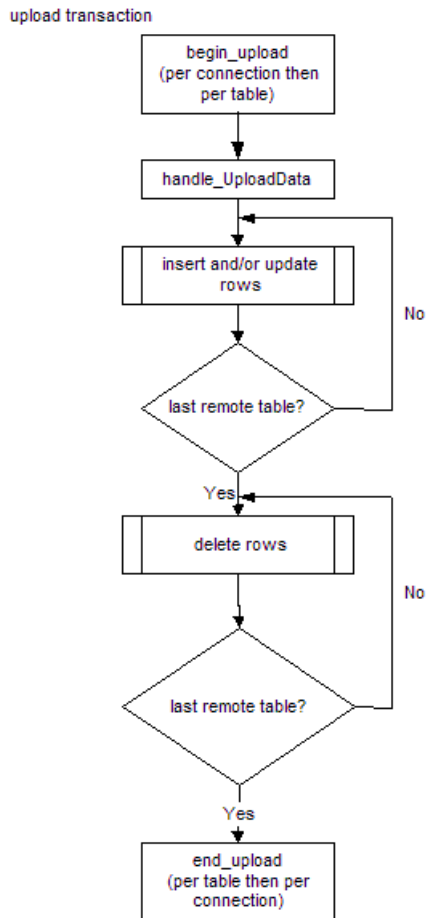
In addition, you can have two connection transactions. A begin connection transaction occurs right after a consolidated database connection is made, and an end connection transaction occurs when the connection is closed.

The primary phases of a synchronization are the upload and download transactions. The events contained in the upload and download transactions are outlined in the following topics.

### **1.12.2.1.2 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.



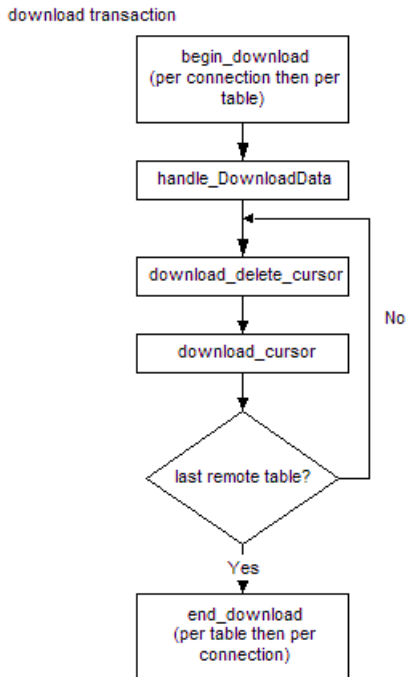
The end\_upload event marks the end of the upload transaction.

You can specify multiple upload transactions with the dbmsync -tu option.

### 1.12.2.1.3 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.



### 1.12.2.1.4 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.

### 1.12.2.1.5 MobiLink Event Model Notes

Keep the following points in mind:

- 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.

## Related Information

[Scripts to Upload Rows \[page 321\]](#)

[Scripts to Download Rows \[page 324\]](#)

### 1.12.2.1.6 MobiLink Complete Event Model

This representation of the MobiLink event model assumes a full synchronization (not upload-only or download-only) with no errors.

The following pseudocode provides an overview of the sequence in which events, and the scripts of the same names, are invoked.

```
-----
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 ) {
  UseDefaultAuthentication <- FALSE
  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( ( not error ) or ( begin_publication script was processed ) ) {
        end_publication
    }
}
for each table being synchronized {
    if( ( not error ) or ( begin_synchronization table script was processed ) ) {
        end_synchronization // Table event.
    }
}
if( ( not error ) or ( begin_synchronization connection script was processed ) )
{
    end_synchronization    // Connection event.
}

```

## Related Information

[Events During Upload \[page 342\]](#)

[Events During Download \[page 345\]](#)

[prepare\\_for\\_download Connection Event \[page 470\]](#)

### 1.12.2.1.7 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.

## Overview of the Upload

```
-----  
upload  
-----  
begin_upload // Connection event  
for each table being synchronized {  
    begin_upload // Table event  
}  
handle_UploadData  
for each table being synchronized {  
    begin_upload_rows  
    for each uploaded INSERT or UPDATE for this table {  
        if( INSERT ) {  
            <upload_inserted_row>  
        }  
        if( UPDATE ) {  
            <upload_updated_row>  
        }  
    }  
    end_upload_rows  
}  
for each table being synchronized IN REVERSE ORDER {  
    begin_upload_deletes  
    for each uploaded DELETE for this table {  
        <upload_deleted_row>  
    }  
    end_upload_deletes  
}  
For each table being synchronized {  
    if( begin_upload table script was processed ) {  
        end_upload // Table event  
    }  
}  
if( begin_upload connection script was processed ) {  
    end_upload // Connection event  
}  
for each table being synchronized {  
    upload_statistics // Table event.  
}  
upload_statistics // Connection event.  
COMMIT
```

## Upload Inserts

```
-----  
<upload_inserted_row>  
-----  
// NOTES:  
// - Only table scripts for the current table are involved.  
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 be 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
}
-----
```

### 1.12.2.1.8 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 the overview of MobiLink events.

```
-----
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.

## Related Information

[Overview of MobiLink Events \[page 336\]](#)  
[Referential Integrity and Synchronization](#)

### 1.12.2.2 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 non-data 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.

```
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();
}
```

For data scripts, in order to upload and download table row data values, all of the Java and .NET events need to be re-written using direct row handling via the `handle_UploadData` and `handle_DownloadData` events.

## Related Information

[Direct Row Handling \[page 558\]](#)

[download\\_cursor Table Event \[page 393\]](#)

[download\\_delete\\_cursor Table Event \[page 396\]](#)

[handle\\_UploadData Connection Event \[page 448\]](#)

[handle\\_DownloadData Connection Event \[page 437\]](#)

[upload\\_delete Table Event \[page 500\]](#)

[upload\\_fetch Table Event \[page 502\]](#)

[upload\\_fetch\\_column\\_conflict Table Event \[page 504\]](#)

[upload\\_insert Table Event \[page 506\]](#)

[upload\\_new\\_row\\_insert Table Event \[page 508\]](#)

[upload\\_old\\_row\\_insert Table Event \[page 510\]](#)

[upload\\_update Table Event \[page 522\]](#)

## 1.12.2.3 authenticate\_file\_transfer Connection Event

Implements custom authentication for file transfers using the mfiletransfer 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.file_authentication_code	<p>INTEGER. Required. This is an INOUT parameter. It indicates the overall success of the authentication.</p> <p>If this value is 1000-1999, file transfer is allowed. If this value is 2000-2999, file transfer is not allowed.</p>	1
s.filename	<p>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 mlsrv17 -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.</p>	2
s.username	<p>VARCHAR(128). The MobiLink user name.</p>	3
s.subdir	<p>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.</p>	Not applicable

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_key	VARCHAR(128). Optional IN parameter to specify a remote key for the file transfer.	Not applicable
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.	Not applicable

## Remarks

The MobiLink server executes this event before allowing any download file transfer using the mlfiletransfer utility or MLFileDownload method. It is executed after the user has authenticated using regular authentication. If this script is not defined, the file transfer is allowed.

The MLFileDownload method can only be used by UltraLite clients.

## Related Information

[Script Additions and Deletions \[page 316\]](#)

[MobiLink File Transfers](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[SQL-Java Data Types \[page 530\]](#)

[-ftr mlsrv17 Option \[page 64\]](#)

[-ftru mlsrv17 Option \[page 65\]](#)

[MobiLink File Transfer Utility \(mlfiletransfer\)](#)

[SQL-.NET Data Types \[page 546\]](#)

### 1.12.2.4 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.file_authentication_code	<p>INTEGER. Required. This is an INOUT parameter. It indicates the overall success of the authentication.</p> <p>If this value is 1000-1999, file transfer is allowed. If this value is 2000-2999, file transfer is not allowed.</p>	1
s.filename	<p>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 mlsrv17 -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.</p>	2
s.file_size	<p>INTEGER. This optional IN parameter can be used to limit the size of file that can be uploaded.</p>	Not applicable
s.username	<p>VARCHAR(128). The MobiLink user name.</p>	3
s.subdir	<p>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.</p>	Not applicable
s.remote_key	<p>VARCHAR(128). Optional IN parameter to specify a remote key for the file transfer.</p>	Not applicable
s.script_version	<p>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.</p>	Not applicable

## Remarks

The MobiLink server executes this event before allowing any download file transfer using the mlfiletransfer utility or MLFileUpload method. It is executed after the user has authenticated using regular authentication. If this script is not defined, the file transfer is allowed.

The MLFileUpload method can only be used by UltraLite clients.

## Related Information

[Script Additions and Deletions \[page 316\]](#)

[MobiLink File Transfers](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[SQL-Java Data Types \[page 530\]](#)

[-ftr mlsrv17 Option \[page 64\]](#)

[-ftru mlsrv17 Option \[page 65\]](#)

[MobiLink File Transfer Utility \(mlfiletransfer\)](#)

[SQL-.NET Data Types \[page 546\]](#)

### 1.12.2.5 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.authentication_status	INTEGER. This is an INOUT parameter.	1
s.authentication_message	VARCHAR(1024). This is an INOUT parameter. Provides an authentication message.	Not applicable

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.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.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
a.N (one or more)	VARCHAR(4000). For example, named parameters could be a . 1 a . 2.	3 or higher

## Parameter Description

### authentication\_status

The authentication\_status parameter is required. It indicates the overall success of the authentication, and can be set to one of the following values:

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

### 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.

**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.

## 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. To prevent the strings from being viewed as plain text, the strings are sent using the same obfuscation as passwords. If `num_auth_parms` is 0, set `auth_parms` to null. The following is an example of passing parameters in UltraLite:

```
ul_char * Params[ 3 ] = { UL_TEXT( "param1" ),
    UL_TEXT( "param2" ), UL_TEXT( "param3" ) };
...
info.num_auth_parms = 3;
info.auth_parms = Params;
```

For SQL Anywhere remote databases, you pass parameters using the `dbmlsync -ap` option, in a comma-separated list. For example, the following command line passes three parameters:

```
dbmlsync -ap "param1,param2,param3"
```

In this example, the `authenticate_parameters` script could be:

```
CALL my_auth_parm (
  {ml s.authentication_status},
  {ml s.remote_id},
  {ml s.username},
  {ml a.1},
  {ml a.2},
  {ml a.3}
)
```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Authentication Parameters \[page 311\]](#)

[MobiLink Users in a Synchronization System](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[Custom User Authentication](#)

[SQL-Java Data Types \[page 530\]](#)

[authenticate\\_user Connection Event \[page 354\]](#)

[authenticate\\_user\\_hashed Connection Event \[page 360\]](#)

[begin\\_synchronization Connection Event \[page 377\]](#)

[-ap dbmlsync Option](#)

[Authentication Parameters Synchronization Parameter](#)

[Number of Authentication Parameters Synchronization Parameter](#)

[SQL-.NET Data Types \[page 546\]](#)

### 1.12.2.6 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.authentication_status	INTEGER. This is an INOUT parameter.	1
s.authentication_message	VARCHAR(1024). This is an INOUT parameter. Provides an authentication message.	Not applicable
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
s.password	VARCHAR(128). The password for authentication purposes. If the user does not supply a password, this value is null.	3
s.new_password	VARCHAR(128). The new password, if this is being used to reset the password. If the user does not change their password, this value is null.	4
s.new_remote_id	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.	
s.new_username	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.	
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.	Not applicable

## Default Action

Use MobiLink built-in user authentication mechanism.

## Remarks

The MobiLink server executes this event upon starting each synchronization. It is executed in a transaction before the begin\_synchronization transaction.

You can use this event to replace the built-in MobiLink authentication mechanism with a custom mechanism. You may want to call into the authentication mechanism of your DBMS, or you may want to implement features not present in the MobiLink built-in mechanism, such as password expiry or a minimum password length.

The parameters used in an `authenticate_user` event are as follows:

#### **authentication\_status**

The `authentication_status` parameter is required. It indicates the overall success of the authentication, and can be set to one of the following values:

Returned Value	<code>authentication_status</code>	Description
$V \leq 1999$	1000	Authentication succeeded.
$2000 \leq V \leq 2999$	2000	Authentication succeeded: password expiring soon.
$3000 \leq V \leq 3999$	3000	Authentication failed: password expired.
$4000 \leq V \leq 4999$	4000	Authentication failed.
$5000 \leq V \leq 5999$	5000	Unable to authenticate because the remote ID is already in use. Try the synchronization again later.
$6000 \leq V$	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.

#### **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 preprocessing 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.

#### **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.

## SQL Example

A typical `authenticate_user` script is a call to a stored procedure. The order of the parameters in the call must match the order above. The following example uses `ml_add_connection_script` to assign the event to a stored procedure called `my_auth`.

```
CALL ml_add_connection_script(
  'ver1', 'authenticate_user', 'call my_auth ( {ml s.authentication_status}, {ml
  s.username} )'
)
```

The following SQL Anywhere stored procedure uses only the user name to authenticate, it has no password check. The procedure ensures only that the supplied user name is one of the employee IDs listed in the `ULEmployee` table.

```
CREATE PROCEDURE my_auth( inout @auth_status int, in @user_name varchar(128) )
BEGIN
  IF EXISTS
    ( SELECT * FROM ulemployee
      WHERE emp_id = @user_name )
  THEN
    MESSAGE 'OK' type info to client;
    SET @auth_status = 1000;
  ELSE
    MESSAGE 'Not OK' type info to client;
    SET @auth_status = 4000;
  END IF
END
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called `authenticateUser` as the script for the `authenticate_user` event when synchronizing the script version `ver1`. This syntax is for SQL Anywhere consolidated databases.

```
CALL ml_add_java_connection_script(
    'ver1', 'authenticate_user',
    'ExamplePackage.ExampleClass.authenticateUser'
)
```

The following is the sample Java method `authenticateUser`. It calls Java methods that check and, if needed, change the user's password.

```
public void authenticateUser(
    com.sap.ml.script.InOutInteger authStatus,
    String user,
    String pwd,
    String newPwd )
    throws java.sql.SQLException {
    // A real authenticate_user handler would
    // handle more authentication code states.
    _curUser = user;
    if( checkPwd( user, pwd ) ) {
        // Authentication successful.
        if( newPwd != null ) {
            // Password is being changed.
            if( changePwd( user, pwd, newPwd ) ) {
                // Authentication OK and password change OK.
                // Use custom code.
                authStatus.setValue( 1001 );
            } else {
                // Authentication OK but password
                // change failed. Use custom code.
                java.lang.System.err.println( "user: "
                    + user + " pwd change failed!" );
                authStatus.setValue( 1002 );
            }
        } else {
            authStatus.setValue( 1000 );
        }
    } else {
        // Authentication failed.
        authStatus.setValue( 4000 );
    }
}
```

## .NET Example

The following call to a MobiLink system procedure registers a .NET method called `AuthUser` as the script for the `authenticate_user` connection event when synchronizing the script version `ver1`. This syntax is for SQL Anywhere consolidated databases.

```
CALL ml_add_dnet_connection_script(
    'ver1', 'authenticate_user',
    'TestScripts.Test.AuthUser'
)
```

The following is the sample .NET method AuthUser. It calls .NET methods that check and, if needed, change the user's password.

```
namespace TestScripts {
public class Test {
    string _curUser = null;
public void AuthUser(
    ref int authStatus,
    string user,
    string pwd,
    string newPwd ) {
    // A real authenticate_user handler would
    // handle more authentication code states.
    _curUser = user;
    if( CheckPwd( user, pwd ) ) {
        // Authentication successful.
        if( newPwd != null ) {
            // Password is being changed.
            if( ChangePwd( user, pwd, newPwd ) ) {
                // Authentication OK and password change OK.
                // Use custom code.
                authStatus = 1001;
            } else {
                // Authentication OK but password
                // change failed. Use custom code.
                System.Console.WriteLine( "user: "
                    + user + " pwd change failed!" );
                authStatus = 1002;
            }
        } else {
            authStatus = 1000 ;
        }
    } else {
        // Authentication failed.
        authStatus = 4000;
    }
}}}
```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[MobiLink Users in a Synchronization System](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[Custom User Authentication](#)

[SQL-Java Data Types \[page 530\]](#)

[authenticate\\_user\\_hashed Connection Event \[page 360\]](#)

[authenticate\\_parameters Connection Event \[page 351\]](#)

[Authentication Value Synchronization Parameter](#)

[sp\\_hook\\_dbmlsync\\_upload\\_end](#)

[modify\\_user Connection Event \[page 463\]](#)

[begin\\_synchronization Connection Event \[page 377\]](#)

[SQL-.NET Data Types \[page 546\]](#)

[.NET Synchronization Example \[page 556\]](#)

## 1.12.2.7 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.authentication_status	INTEGER. This is an INOUT parameter.	1
s.authentication_message	VARCHAR(1024). This is an INOUT parameter. Provides an authentication message.	Not applicable
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
s.hashed_password	BINARY(32). If the user does not supply a password, this value is null.	3
s.hashed_new_password	BINARY(32). If this event is not being used to change the user's password, this value is null.	4
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.	Not applicable

### Default Action

Use MobiLink built-in user authentication mechanism.



## Remarks

This event is identical to `authenticate_user` except for the passwords, which are in the same hashed form as those stored in the `ml_user.hashed_password` column. Passing the passwords in hashed form provides increased security.

A one-way hash is used. A one-way hash takes a password and converts it to a byte sequence that is (essentially) unique to each possible password. The one-way hash lets password authentication take place without having to store the actual password in the consolidated database.

Due to incremental improvements in the quality of the hash across MobiLink versions, this script can be called multiple times during an authentication sequence for a user.

When `authenticate_user` and `authenticate_user_hashed` are both defined, and both scripts return different `authentication_status` codes, the higher value is used.

## SQL Example

A typical `authenticate_user_hashed` script is a call to a stored procedure. The order of the parameters in the call must match the order above. The following example calls `ml_add_connection_script` to assign the event to a stored procedure called `my_auth`.

```
CALL ml_add_connection_script(
  'ver1', 'authenticate_user_hashed',
  'call my_auth (
    {ml s.authentication_status},
    {ml s.username},
    {ml s.hashed_password})'
)
```

The following SQL Anywhere stored procedure uses both the user name and password to authenticate. The procedure ensures only that the supplied user name is one of the employee IDs listed in the `ULEmployee` table. The procedure assumes that the `Employee` table has a `binary(20)` column called `hashed_pwd`.

```
CREATE PROCEDURE my_auth(
  inout @authentication_status integer,
  in @user_name varchar(128),
  in @hpwd binary(32) )
BEGIN
  IF EXISTS
  ( SELECT * FROM ulemmployee
    WHERE emp_id = @user_name
      and hashed_pwd = @hpwd )
  THEN
    message 'OK' type info to client;
    RETURN 1000;
  ELSE
    message 'Not OK' type info to client;
    RETURN 4000;
  END IF
END
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called `authUserHashed` as the script for the `authenticate_user_hashed` event when synchronizing the script version `ver1`.

```
CALL ml_add_java_connection_script(  
    'ver1', 'authenticate_user_hashed',  
    'ExamplePackage.ExampleClass.authUserHashed')
```

The following is the sample Java method `authUserHashed`. It calls Java methods that check and, if needed, change the user's password.

```
public void authUserHashed(  
    com.sap.ml.script.InOutInteger authStatus,  
    String user,  
    byte pwd[],  
    byte newPwd[] )  
    throws java.sql.SQLException {  
    // A real authenticate_user_hashed handler  
    // would handle more auth code states.  
    _curUser = user;  
    if( checkPwdHashed( user, pwd ) ) {  
        // Authorization successful.  
        if( newPwd != null ) {  
            // Password is being changed.  
            if( changePwdHashed( user, pwd, newPwd ) ) {  
                // Authorization OK and password change OK.  
                // Use custom code.  
                authStatus.setValue( 1001 );  
            } else {  
                // Auth OK but password change failed.  
                // Use custom code  
                java.lang.System.err.println( "user: " + user  
                    + " pwd change failed!" );  
                authStatus.setValue( 1002 );  
            }  
        } else {  
            authStatus.setValue( 1000 );  
        }  
    } else {  
        // Authorization failed.  
        authStatus.setValue( 4000 );  
    }  
}
```

## .NET Example

The following call to a MobiLink system procedure registers a .NET method called `AuthUserHashed` as the script for the `authenticate_user_hashed` connection event when synchronizing the script version `ver1`. This syntax is for SQL Anywhere consolidated databases.

```
CALL ml_add_dnet_connection_script(  
    'ver1',  
    'authenticate_user_hashed',  
    'TestScripts.Test.AuthUserHashed'  
)
```

The following is the sample .NET method AuthUserHashed.

```
namespace TestScripts {
public class Test {
    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( CheckPwdHashed( user, pwdHash ) ) {
        // Authorization successful.
        if( newPwdHash != null ) {
            // Password is being changed.
            if( ChangePwdHashed( user, pwdHash, newPwdHash ) ) {
                // Authorization OK and password change OK.
                // Use custom code.
                authStatus = 1001;
            } else {
                // Auth OK but password change failed.
                // Use custom code
                System.Console.WriteLine( "user: " + user
                    + " pwd change failed!" );
                authStatus = 1002;
            }
        } else {
            authStatus = 1000;
        }
    } else {
        // Authorization failed.
        authStatus = 4000;
    }
}}}
```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[MobiLink Users in a Synchronization System](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[Custom User Authentication](#)

[SQL-Java Data Types \[page 530\]](#)

[authenticate\\_user Connection Event \[page 354\]](#)

[authenticate\\_parameters Connection Event \[page 351\]](#)

[SQL-.NET Data Types \[page 546\]](#)

## 1.12.2.8 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.

#### **i** 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.

### SQL Example

The following SQL script works with a SQL Anywhere consolidated database. Two variables are created, one for the last\_download timestamp, and one for employee ID.

```
CALL ml_add_connection_script(  
    'custdb',  
    'begin_connection',  
    'create variable @LastDownload timestamp;  
    create variable @EmployeeID integer;')
```

### Related Information

[Script Additions and Deletions \[page 316\]](#)

[end\\_connection Connection Event \[page 405\]](#)

[-cn mlsrv17 Option \[page 55\]](#)

[-w mlsrv17 Option \[page 96\]](#)

## 1.12.2.9 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 retry the script (since it cannot roll it back).

This event only executes if a script has been defined for the event.

### Related Information

[Script Additions and Deletions \[page 316\]](#)

## 1.12.2.10 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.last_download	TIMESTAMP. The oldest download time of any synchronized table.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
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.	Not applicable

### 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.

## 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 );
}
```

## Related Information

[Script Parameters \[page 294\]](#)  
[Script Additions and Deletions \[page 316\]](#)  
[Remote IDs and MobiLink User Names in Scripts](#)  
[Last Download Times in Scripts \[page 117\]](#)  
[SQL-Java Data Types \[page 530\]](#)  
[end\\_download Connection Event \[page 407\]](#)  
[SQL-.NET Data Types \[page 546\]](#)

### 1.12.2.11 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.last_table_download	TIMESTAMP. The last download time for the table.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
s.table	VARCHAR(128). The table name.	3
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.	Not applicable



## 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.

## 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.

```
CALL ml_add_table_script(
    'version1',
    'Leads',
    'begin_download',
    'CALL BeginTableDownload(
        {ml s.username},
        {ml s.table} )' );
```

The following SQL statements create the `BeginTableDownload` procedure. It records the download attempt in a table.

```
CREATE PROCEDURE BeginTableDownload(
    MLUser varchar(128),
    TableName varchar(128) )
BEGIN
    INSERT INTO DownloadAttempts ( MLUser, TableName, LastDownload );
END
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called `beginDownloadTable` as the script for the `begin_download` table event when synchronizing the script version `ver1`.

```
CALL ml_add_java_table_script(
    'ver1',
    'table1',
    'begin_download',
    'ExamplePackage.ExampleClass.beginDownloadTable'
)
```

The following is the sample Java method `beginDownloadTable`. It prints a message to the MobiLink message log. (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 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`.

```
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.)

```
namespace TestScripts {
public class Test {
    string _curUser = null;
public void BeginTableDownload(
    string user,
    string table ) {
    System.Console.WriteLine("Beginning to process download for: " + table);
}}}
```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[Last Download Times in Scripts \[page 117\]](#)

[SQL-Java Data Types \[page 530\]](#)

[end\\_download Table Event \[page 409\]](#)

[SQL-.NET Data Types \[page 546\]](#)

## 1.12.2.12 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.last_table_download	TIMESTAMP. The last download time for the table.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
s.table	VARCHAR(128). The table name.	3
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.	Not applicable

### 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.

## i 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.

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[Last Download Times in Scripts \[page 117\]](#)

[SQL-Java Data Types \[page 530\]](#)

[begin\\_download\\_rows Table Event \[page 372\]](#)

[end\\_download\\_rows Table Event \[page 413\]](#)

[SQL-.NET Data Types \[page 546\]](#)

## 1.12.2.13 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
<code>s.last_table_download</code>	TIMESTAMP. The last download time for 1 the table.	1

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
s.table	VARCHAR(128). The table name.	3
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.	Not applicable

## 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.

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[Last Download Times in Scripts \[page 117\]](#)

[SQL-Java Data Types \[page 530\]](#)

[begin\\_download\\_deletes Table Event \[page 371\]](#)

## 1.12.2.14 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.generation_number	INTEGER. This is an INOUT parameter. If your deployment does not use file-based downloads, this parameter can be ignored. The default is 1.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
s.publication_name	VARCHAR(128). The name of the publication.	3
s.last_publication_upload	TIMESTAMP. The time of the last successful upload of this publication.	4
s.last_publication_download	TIMESTAMP. The last download time for the publication.	5
s.subscription_id	VARCHAR(128). The remote subscription ID.	6

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
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.	Not applicable

## 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.

## SQL Example

You may want to record the information for each publication being synchronized. The following example calls `ml_add_connection_script` to assign the event to a stored procedure called `RecordPubSync`.

```
CALL ml_add_connection_script(  
  'version1',  
  'begin_publication',  
  '{CALL RecordPubSync(  
    {ml s.generation_number},  
    {ml s.username},  
    {ml s.publication_name},  
    {ml s.last_publication_upload},  
    {ml s.last_publication_download},  
    {ml s.subscription_id}})' );
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called `beginPublication` as the script for the `begin_publication` connection event when synchronizing the script version `ver1`.

```
CALL ml_add_java_connection_script(  
  'ver1',  
  'begin_publication',  
  'ExamplePackage.ExampleClass.beginPublication' )
```

The following is the sample Java method `beginPublication`. It saves the name of each publication for later use.

```
package ExamplePackage;  
public class ExampleClass  
{  
  java.util.ArrayList<String> _publicationNames;  
  int _numPublications = 0;  
  public void beginPublication( com.sap.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 );
        }
    }
}
```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[MobiLink File-based Download \[page 270\]](#)

[MobiLink Generation Numbers \[page 277\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[Last Download Times in Scripts \[page 117\]](#)

[SQL-Java Data Types \[page 530\]](#)

[end\\_publication Connection Event \[page 415\]](#)

[SQL-.NET Data Types \[page 546\]](#)

### 1.12.2.15 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.new_remote_id	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.	
s.new_username	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.	
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.	Not applicable

## 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.

## 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.

```
CALL ml_add_connection_script (
    'version1',
    'begin_synchronization',
    'set @EmployeeID = {ml s.username}' );
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called `beginSynchronizationConnection` as the script for the `begin_synchronization` connection event when synchronizing the script version `ver1`.

```
CALL ml_add_java_connection_script(  
  'ver1',  
  'begin_synchronization',  
  'ExamplePackage.ExampleClass.beginSynchronizationConnection'  
)
```

The following is the sample Java method `beginSynchronizationConnection`. It saves the name of the synchronizing user for later use.

```
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`.

```
CALL ml_add_dnet_connection_script( 'ver1',  
  'begin_synchronization',  
  'TestScripts.Test.BeginSync'  
)
```

The following is the sample .NET method `BeginSync`. It saves the name of the synchronizing user for later use.

```
namespace TestScripts {  
    public class Test {  
        string _curUser = null;  
        public void BeginSync(  
            string user ) {  
            _curUser = user;  
        }  
    }  
}
```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[SQL-Java Data Types \[page 530\]](#)

[end\\_synchronization Connection Event \[page 418\]](#)

## 1.12.2.16 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2
s.new_remote_id	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.	
s.new_username	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.	
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.	Not applicable

## 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.

## SQL Example

The `begin_synchronization` table event is used to set up the synchronization of a particular table. The following SQL script registers a script that creates a temporary table for storing rows during synchronization. This syntax is for a SQL Anywhere consolidated database.

```
CALL ml_add_table_script(
  'ver1',
  'sales_order',
  'begin_synchronization',
  'CREATE TABLE #sales_order (
    id          integer NOT NULL default autoincrement,
    cust_id     integer NOT NULL,
    order_date  date NOT NULL,
    fin_code_id char(2) NULL,
    region      char(7) NULL,
    sales_rep   integer NOT NULL,
    PRIMARY KEY (id),
  )' )
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called `beginSynchronizationTable` as the script for the `begin_synchronization` table event when synchronizing the script version `ver1`.

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'begin_synchronization',
  'ExamplePackage.ExampleClass.beginSynchronizationTable' )
```

The following is the sample Java method `beginSynchronizationTable`. It adds the current table name to a list of table names contained in this instance.

```
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.

```

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.

```

using System.Collections.Generic;
using System;
namespace TestScripts
{
    class Test
    {
        List<string> _tableList = new List<string>();
        string _curTable = "";
        public void BeginSynchronizationTable( string user,
                                             string table )
        {
            _curTable = table;
            _tableList.Add( table );
        }
        public void EndTableDownload( DateTime timestamp,
                                     string user,
                                     string table )
        {
            _curTable = null;
        }
    }
}

```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[SQL-Java Data Types \[page 530\]](#)

[end\\_synchronization Table Event \[page 421\]](#)

[begin\\_synchronization Connection Event \[page 377\]](#)

[SQL-.NET Data Types \[page 546\]](#)

### 1.12.2.17 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.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.	Not applicable

## 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.

## SQL Example

The `begin_upload` connection event is used to perform whatever steps you need performed before uploading rows. The following SQL script creates a temporary table for storing old and new row values for conflict processing of the `sales_order` table. This example works with a SQL Anywhere consolidated database.

```
CALL ml_add_connection_script(
  'version1',
  'begin_upload',
  'CREATE TABLE #sales_order_conflicts (
    id          integer NOT NULL default autoincrement,
    cust_id     integer NOT NULL,
    order_date  date NOT NULL,
    fin_code_id char(2) NULL,
    region      char(7) NULL,
    sales_rep   integer NOT NULL,
    new_value   char(1) NOT NULL,
    PRIMARY KEY (id) )' )
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called `beginUploadConnection` as the script for the `begin_upload` connection event when synchronizing the script version `ver1`.

```
CALL ml_add_java_connection_script(
  'ver1',
  'begin_upload',
  'ExamplePackage.ExampleClass.beginUploadConnection ' )
```

The following is the sample Java method `beginUploadConnection`. It prints a message to the MobiLink message log. (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 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.

```
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.

```
namespace TestScripts {  
    public class Test {  
        string _curUser = null;  
        public void BeginUpload( string curUser ) {  
            _curUser = curUser;  
        }  
    }  
}
```

## Related Information

- [Script Parameters \[page 294\]](#)
- [Script Additions and Deletions \[page 316\]](#)
- [Remote IDs and MobiLink User Names in Scripts](#)
- [SQL-Java Data Types \[page 530\]](#)
- [end\\_upload Connection Event \[page 424\]](#)
- [begin\\_upload Table Event \[page 385\]](#)
- [SQL-.NET Data Types \[page 546\]](#)

### 1.12.2.18 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2
s.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.	Not applicable

## 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 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.

## 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.

```
CALL ml_add_table_script(
  'version1',
  'Leads',
  'begin_upload',
```

```
'DELETE FROM T_Leads' )
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called `beginUploadTable` as the script for the `begin_upload` table event when synchronizing the script version `ver1`.

```
CALL ml_add_java_table_script(  
  'ver1',  
  'table1',  
  'begin_upload',  
  'ExamplePackage.ExampleClass.beginUploadTable'  
)
```

The following is the sample Java method `beginUploadTable`. It prints a message to the MobiLink message log. (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 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`.

```
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.)

```
namespace TestScripts {  
  public class Test {  
    string _curUser = null;  
    public void BeginTableUpload(  
      string user,  
      string table ) {  
      System.Console.WriteLine("Beginning to process upload for: " + table);  
    }  
  }  
}
```

## Related Information

[Script Parameters \[page 294\]](#)  
[Script Additions and Deletions \[page 316\]](#)  
[Remote IDs and MobiLink User Names in Scripts](#)  
[SQL-Java Data Types \[page 530\]](#)  
[end\\_upload Table Event \[page 426\]](#)  
[begin\\_upload Connection Event \[page 383\]](#)  
[SQL-.NET Data Types \[page 546\]](#)

### 1.12.2.19 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2
s.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.	Not applicable

## 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.

## SQL Example

The `begin_upload_deletes` table event is used to perform whatever steps you need performed after uploading inserts and updates for a particular table, but before deletes are uploaded for that table. The following SQL script creates a temporary table for storing deletes temporarily during upload. This syntax is for a SQL Anywhere consolidated database.

```
CALL ml_add_table_script(
  'ver1',
  'sales_order',
  'begin_upload_deletes',
  'CREATE TABLE #sales_order_deletes (
    id          integer NOT NULL default autoincrement,
    cust_id     integer NOT NULL,
    order_date  date NOT NULL,
    fin_code_id char(2) NULL,
    region      char(7) NULL,
    sales_rep   integer NOT NULL,
    PRIMARY KEY (id) )' )
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called `beginUploadDeletes` as the script for the `begin_upload_deletes` table event when synchronizing the script version `ver1`.

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'begin_upload_deletes',
  'ExamplePackage.ExampleClass.beginUploadDeletes' )
```

The following is the sample Java method `beginUploadDeletes`. It prints a message to the MobiLink message log. (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 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`.

```
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.)

```
namespace TestScripts {
public class Test {
    string _curUser = null;
public void BeginUploadDeletes(
    string user,
    string table ) {
    System.Console.WriteLine(
    "Starting upload deletes for table: " + table );
}}

```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[SQL-Java Data Types \[page 530\]](#)

[end\\_upload\\_deletes Table Event \[page 429\]](#)

[SQL-.NET Data Types \[page 546\]](#)

## 1.12.2.20 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2
s.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.	Not applicable

### 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.

## 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.)

```
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 );
    }}
}
```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[SQL-Java Data Types \[page 530\]](#)

[end\\_upload\\_rows Table Event \[page 432\]](#)

[SQL-.NET Data Types \[page 546\]](#)

### 1.12.2.21 download\_cursor Table Event

A data script that defines a cursor to select rows to download and insert and 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. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.last_table_download	TIMESTAMP. The last download time for the table.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.username	VARCHAR(128). The MobiLink user name.	2
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.	Not applicable

## Default Action

None.

## Remarks

The MobiLink server uses the script to open a read-only cursor to fetch a list of rows to download to the remote database.

You can have one `download_cursor` script for each table in the remote database.

To optimize performance of the download stage of synchronization to UltraLite clients, when the range of primary key values is outside the current rows on the device, you should order the rows in the download cursor by primary key. Downloads of large reference tables, for example, can benefit from this optimization.

Each `download_cursor` script must contain a `SELECT` statement or a call to a procedure that returns a result set. The MobiLink server uses this statement to define a cursor in the consolidated database.

The script must select all columns that correspond to the columns in the corresponding table in the remote database. The columns in the consolidated database can have different names than the corresponding columns in the remote database, but they must be of compatible types.

The columns must be selected in the order that the corresponding columns are defined in the remote database.

To avoid downloading unnecessary rows, consider using timestamp-based downloads. When using timestamp-based downloads, include a line similar to the following in the `WHERE` clause of your `download_cursor` script:

```
AND last_modified >= {ml s.last_table_download}
```

This script must be implemented in SQL. For Java or .NET processing of rows, use direct row handling.

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.

## SQL Example

The following example comes from an Oracle installation, although the statement is valid against all supported databases. This example downloads all rows that have been changed since the last time the user downloaded data, and that match the user name in the emp\_name column.

```
CALL ml_add_table_script(  
  'Lab',  
  'ULOrder',  
  'download_cursor',  
  'SELECT order_id,  
    cust_id,  
    prod_id,  
    emp_id,  
    disc,  
    quant,  
    notes,  
    status  
  FROM ULOrder  
  WHERE last_modified >= {ml s.last_table_download}  
    AND emp_name = {ml s.username}' )
```

## Related Information

[Direct Row Handling \[page 558\]](#)

[Data Scripts \[page 346\]](#)

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Scripts to Download Rows \[page 324\]](#)

[download\\_cursor Scripts \[page 326\]](#)

[Partitioned Rows Among Remote Databases \[page 121\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[Last Download Times in Scripts \[page 117\]](#)

## 1.12.2.22 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. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.last_table_download	TIMESTAMP. The last download time for the table.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
s.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.	Not applicable

### 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.

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:

```
AND last_modified >= {ml s.last_table_download}
```

This script must be implemented in SQL. For Java or .NET processing of rows, use Direct row handling.

It can be problematic using `READPAST` table hints in a `download_delete_cursor`. For details, see the `download_cursor` event.

## 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 )')
```

## Related Information

[Data Scripts \[page 346\]](#)

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Scripts to Download Rows \[page 324\]](#)

[Partitioned Rows Among Remote Databases \[page 121\]](#)

[download\\_delete\\_cursor Scripts \[page 327\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[Last Download Times in Scripts \[page 117\]](#)

[Direct Row Handling \[page 558\]](#)

[download\\_cursor Table Event \[page 393\]](#)

[FROM Clause](#)

## 1.12.2.23 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.

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. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name as specified in your SYNCHRONIZATION USER definition.	1
s.warnings	INTEGER. The number of warnings issued.	2
s.errors	INTEGER. The number of errors, including handled errors, that occurred.	3
s.fetched_rows	INTEGER. The number of rows fetched by the download_cursor script.	4
s.deleted_rows	INTEGER. The number of rows fetched by the download_delete_cursor script.	5
s.filtered_rows	INTEGER. The number of rows from the fetched_rows parameter actually sent to the remote. This reflects download filtering of uploaded values.	6

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.bytes	INTEGER. The number of bytes sent to the remote database as the download.	7
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.	Not applicable

## 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.

### i 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.

## SQL Example

The following example inserts synchronization statistics into a table called `download_audit`.

```
CALL ml_add_connection_script(
  'ver1',
  'download_statistics',
  'INSERT INTO download_audit(
    user_name,
    warnings,
    errors,
    fetched_rows,
    deleted_rows,
    filtered_rows,
    bytes )
  VALUES (
    {ml s.username},
    {ml s.warnings},
    {ml s.errors},
    {ml s.fetched_rows},
    {ml s.deleted_rows},
```

```
{ml s.filtered_rows},
{ml s.bytes})')
```

Once vital statistics are inserted into the audit table, you may use these statistics to monitor your synchronizations and make optimizations where applicable.

## Java Example

The following call to a MobiLink system procedure registers a Java method called `downloadStatisticsConnection` as the script for the `download_statistics` event when synchronizing the script version `ver1`.

```
CALL ml_add_java_connection_script(
  'ver1',
  'download_statistics',
  'ExamplePackage.ExampleClass.downloadStatisticsConnection' )
```

The following is the sample Java method `downloadStatisticsConnection`. It prints the number of fetched rows to the MobiLink message log. (Printing the number of fetched rows 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 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`.

```
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.)

```
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 );
    }}

```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[MobiLink Profiler \[page 247\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[SQL-Java Data Types \[page 530\]](#)

[download\\_statistics Table Event \[page 401\]](#)

[upload\\_statistics Connection Event \[page 513\]](#)

[upload\\_statistics Table Event \[page 517\]](#)

[synchronization\\_statistics Connection Event \[page 487\]](#)

[synchronization\\_statistics Table Event \[page 491\]](#)

[time\\_statistics Connection Event \[page 494\]](#)

[time\\_statistics Table Event \[page 497\]](#)

[SQL-.NET Data Types \[page 546\]](#)

### 1.12.2.24 download\_statistics Table Event

Provides access to synchronization statistics for download operations by table.

#### Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, use the appropriate corresponding data type.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name as specified in your SYNCHRONIZATION USER definition.	1
s.table	VARCHAR(128). The table name.	2
s.warnings	INTEGER. The number of warnings issued.	3
s.errors	INTEGER. The number of errors, including handled errors, that occurred.	4
s.fetched_rows	INTEGER. The number of rows fetched by the download_cursor script.	5
s.deleted_rows	INTEGER. The number of rows fetched by the download_delete_cursor script.	6
s.filtered_rows	INTEGER. The number of rows filtered from the fetched_rows. This reflects download filtering of uploaded values.	7
s.bytes	INTEGER. The number of bytes sent to the remote database as the download.	8
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.	Not applicable

## 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.

## i 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.

## SQL Example

The following example inserts synchronization statistics into a table called `download_audit`. Once vital statistics are inserted into the audit table, you may use these statistics to monitor your synchronizations and make optimizations where applicable.

```
CALL ml_add_table_script(
  'ver1',
  'table1',
  'download_statistics',
  'INSERT INTO download_audit (
    user_name,
    table, warnings,
    errors,
    fetched_rows,
    deleted_rows,
    filtered_rows,
    bytes)
VALUES (
  {ml s.username},
  {ml s.table},
  {ml s.warnings},
  {ml s.errors},
  {ml s.fetched_rows},
  {ml s.deleted_rows},
  {ml s.filtered_rows},
  {ml s.bytes})')
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called `downloadStatisticsTable` as the script for the `download_statistics` table event when synchronizing the script version `ver1`.

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'download_statistics',
  'ExamplePackage.ExampleClass.downloadStatisticsTable' )
```

The following is the sample Java method `downloadStatisticsTable`. It prints some statistics for this table to the MobiLink message log. (Printing statistics for a table 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 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.)

```

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,
int bytes ) {
System.Console.WriteLine( "download table stats "
+ "table: " + table + "bytes: " + bytes );
}}}

```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[MobiLink Profiler \[page 247\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[SQL-Java Data Types \[page 530\]](#)

[download\\_statistics Connection Event \[page 398\]](#)

[upload\\_statistics Connection Event \[page 513\]](#)  
[upload\\_statistics Table Event \[page 517\]](#)  
[synchronization\\_statistics Connection Event \[page 487\]](#)  
[synchronization\\_statistics Table Event \[page 491\]](#)  
[time\\_statistics Connection Event \[page 494\]](#)  
[time\\_statistics Table Event \[page 497\]](#)  
[SQL-.NET Data Types \[page 546\]](#)

## 1.12.2.25 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.

### SQL Example

The following SQL script drops a temporary table that was created by the `begin_connection` script. This syntax is for a SQL Anywhere consolidated database. Strictly speaking, this table doesn't need to be dropped explicitly, since SQL Anywhere does this automatically when the connection is destroyed. Whether a temporary table needs to be dropped explicitly depends on your consolidated database type.

```
CALL ml_add_connection_script(  
    'version 1.0',
```

```
'end_connection',  
'DROP TABLE #sync_info' )
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called endConnection as the script for the end\_connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(  
    'ver1',  
    'end_connection',  
    'ExamplePackage.ExampleClass.endConnection' )
```

The following is the sample Java method endConnection. It prints a message to the MobiLink message log. (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 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.

```
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.)

```
namespace TestScripts {  
    public class Test {  
        string _curUser = null;  
        public void EndConnection() {  
            System.Console.WriteLine( "Ending connection." );  
        }  
    }  
}
```

## Related Information

[Script Additions and Deletions \[page 316\]](#)

## 1.12.2.26 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.last_download	TIMESTAMP. The oldest download time of any synchronized table.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
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.	Not applicable

### 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.

## SQL Example

The following example shows one possible use of an end\_download connection script. This script deletes rows from a temporary table used to help generate the download.

```
CALL ml_add_connection_script(  
  'ver1',  
  'end_download',  
  'DELETE FROM TempDownloadTable where user = {ml s.username}')
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called endDownloadConnection as the script for the end\_download connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(  
  'ver1',  
  'end_download',  
  'ExamplePackage.ExampleClass.endDownloadConnection' )
```

The following is the sample Java method endDownloadConnection. It prints a message to the MobiLink message log. (Printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```
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.

```
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.)

```
public void EndDownload(  
    DateTime timestamp,
```



```
string user ) {
System.Console.WriteLine( "Ending download for user: " + user );
}
```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[Last Download Times in Scripts \[page 117\]](#)

[SQL-Java Data Types \[page 530\]](#)

[begin\\_download Connection Event \[page 366\]](#)

[SQL-.NET Data Types \[page 546\]](#)

## 1.12.2.27 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.last_table_download	TIMESTAMP. The last download time for the table.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
s.table	VARCHAR(128). The table name.	3

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
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.	Not applicable

## 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.

## SQL Example

The end\_download table event is used to perform whatever steps you need performed after downloading a particular table. The following SQL Anywhere SQL script drops a temporary table created by a prepare\_for\_download script to hold download rows for the sales\_summary table.

```
CALL ml_add_table_script(
  'MyCorp 1.0',
  'sales_summary',
  'end_download',
  'DROP TABLE #sales_summary_download' )
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called endDownloadTable as the script for the end\_download table event when synchronizing the script version ver1.

```
CALL ml_add_java_table_script (
  'ver1',
  'table1',
  'end_download',
  'ExamplePackage.ExampleClass.endDownloadTable' )
```

The following is the sample Java method `endDownloadTable`. It resets the current table member variable.

```
public 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`.

```
CALL ml_add_dnet_table_script(
    'ver1',
    'table1',
    'end_download',
    'TestScripts.Test.EndTableDownload'
)
```

The following is the sample .NET method `EndTableDownload`. It resets the current table member variable.

```
public void EndTableDownload
    DateTime timestamp,
    string user,
    string table ) {
    _curTable = null;
}}
```

## Related Information

- [Script Parameters \[page 294\]](#)
- [Script Additions and Deletions \[page 316\]](#)
- [Remote IDs and MobiLink User Names in Scripts](#)
- [Last Download Times in Scripts \[page 117\]](#)
- [SQL-Java Data Types \[page 530\]](#)
- [begin\\_download Table Event \[page 368\]](#)
- [end\\_download Connection Event \[page 407\]](#)
- [SQL-.NET Data Types \[page 546\]](#)

## 1.12.2.28 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.last_table_download	TIMESTAMP. The last download time for the table.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
s.table	VARCHAR(128). The table name.	3
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.	Not applicable

### 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.

## i Note

For each download table, the `begin_download_deletes`, `download_delete_cursor`, and `end_download_deletes` events are invoked in sequence. Consider implementing all of the download delete logic for a table in a `download_delete_cursor` event implemented as a single stored procedure that returns a result set containing all of the rows to be deleted from the remote table. The reduced number of script invocations may result in improved download performance.

You can have one `end_download_deletes` script for each table in the remote database.

## Related Information

[Script Parameters \[page 294\]](#)  
[Script Additions and Deletions \[page 316\]](#)  
[Remote IDs and MobiLink User Names in Scripts](#)  
[Last Download Times in Scripts \[page 117\]](#)  
[SQL-Java Data Types \[page 530\]](#)  
[begin\\_download\\_deletes Table Event \[page 371\]](#)  
[end\\_download Connection Event \[page 407\]](#)  
[begin\\_download\\_rows Table Event \[page 372\]](#)  
[end\\_download\\_rows Table Event \[page 413\]](#)  
[download\\_delete\\_cursor Table Event \[page 396\]](#)  
[SQL-.NET Data Types \[page 546\]](#)

## 1.12.2.29 end\_download\_rows Table Event

Processes statements related to a specific table just after preparing a list of rows to be inserted or updated in the specified table in the remote database.

### Parameters

In the following table, the description provides the SQL data type. If you are writing your script in Java or .NET, use the appropriate corresponding data type.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.last_table_download	TIMESTAMP. The last download time for the table.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
s.table	VARCHAR(128). The table name.	3
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.	Not applicable

## 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.

### **i** 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.

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[Last Download Times in Scripts \[page 117\]](#)  
[SQL-Java Data Types \[page 530\]](#)  
[begin\\_download\\_rows Table Event \[page 372\]](#)  
[end\\_download Connection Event \[page 407\]](#)  
[end\\_download\\_deletes Table Event \[page 412\]](#)  
[begin\\_download\\_deletes Table Event \[page 371\]](#)  
[SQL-.NET Data Types \[page 546\]](#)

## 1.12.2.30 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.generation_number	INTEGER. If your deployment does not use file-based downloads, this parameter can be ignored. The default value is 1.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
s.publication_name	VARCHAR(128). The name of the publication.	3
s.last_publication_upload	TIMESTAMP. Last successful upload time of this publication.	4
s.last_publication_download	TIMESTAMP. The last download time of this publication.	5

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.subscription_id	VARCHAR(128). The remote subscription ID.	6
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.	Not applicable

## 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.



## SQL Example

You may want to record the information for each publication being synchronized. The following example calls `ml_add_connection_script` to assign the event to a stored procedure called `RecordPubEndSync`.

```
CALL ml_add_connection_script(
  'version1',
  'end_publication',
  'CALL RecordPubEndSync(
    {ml s.generation_number},
    {ml s.username},
    {ml s.publication_name},
    {ml s.last_publication_upload},
    {ml s.last_publication_download} )' );
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called `endPublication` as the script for the `end_publication` connection event when synchronizing the script version `ver1`.

```
CALL ml_add_java_connection_script(
  'ver1',
  'end_publication',
  'ExamplePackage.ExampleClass.endPublication' )
```

The following is the sample Java method `endPublication`. It outputs a message to the MobiLink message log. (Printing a message to the MobiLink message log might be useful at development time but would slow down a production server.)

```
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`.

```
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.)

```
public void EndPub(
    int generation_number,
    string user,
    string pub_name,
    DateTime last_publication_upload,
    DateTime last_publication_download ) {
    System.Console.Write(
        "Finished synchronizing publication " + pub_name );
}
```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[MobiLink File-based Download \[page 270\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[Last Download Times in Scripts \[page 117\]](#)

[SQL-Java Data Types \[page 530\]](#)

[begin\\_publication Connection Event \[page 374\]](#)

[SQL-.NET Data Types \[page 546\]](#)

### 1.12.2.31 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.synchronization_ok	INTEGER. This value is 1 for a successful synchronization and 0 for an unsuccessful synchronization.	2
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.	Not applicable

## 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.

## 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`.

```
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.

```
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 + "'");  
      }  
    }  
  }
```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[SQL-Java Data Types \[page 530\]](#)

[begin\\_synchronization Connection Event \[page 377\]](#)

[begin\\_synchronization Table Event \[page 380\]](#)

[end\\_synchronization Table Event \[page 421\]](#)

[SQL-.NET Data Types \[page 546\]](#)

## 1.12.2.32 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2
s.synchronization_ok	INTEGER. This value is 1 for a successful synchronization and 0 for an unsuccessful synchronization.	3
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.	Not applicable

## 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.

## SQL Example

The following SQL Anywhere SQL script drops a temporary table created by the begin\_synchronization script.

```
CALL ml_add_table_script(  
  'ver1',  
  'sales_order',  
  'end_synchronization',  
  'DROP TABLE #sales_order' )
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called endSynchronizationTable as the script for the end\_synchronization table event when synchronizing the script version ver1.

```
CALL ml_add_java_table_script(  
  'ver1',  
  'table1',  
  'end_synchronization',  
  'ExamplePackage.ExampleClass.endSynchronizationTable' )
```

The following is the sample Java method endSynchronizationTable.

```
package ExamplePackage;  
import com.sap.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.

```
using Sap.MobiLink.Script;  
namespace TestScripts {  
  public class ExampleClass {  
    DBConnectionContext _cc = null;  
    ExampleClass( DBConnectionContext cc ) {  
      _cc = cc;  
    }  
    public void EndSynchronizationTable() {  
      DBConnection conn = _cc.GetConnection();  
      try {  
        DBCommand cmd = conn.CreateCommand();  
        try {  
          cmd.CommandText = "DROP TABLE #sales_order";  
          cmd.Prepare();  
          cmd.ExecuteNonQuery();  
        } finally {  
          cmd.Close();  
        }  
      } finally {  
        conn.Close();  
      }  
    }  
  }  
}
```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[SQL-Java Data Types \[page 530\]](#)

[begin\\_synchronization Table Event \[page 380\]](#)

[end\\_synchronization Connection Event \[page 418\]](#)

[SQL-.NET Data Types \[page 546\]](#)

## 1.12.2.33 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.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.	Not applicable

### 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.



## SQL Example

The following SQL Anywhere SQL script calls the EndUpload stored procedure.

```
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.

```
CALL ml_add_java_connection_script(  
    'ver1',  
    'end_upload',  
    'ExamplePackage.ExampleClass.endUploadConnection' )
```

The following is the sample Java method endUploadConnection. It calls a method to perform operations on the database.

```
public 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.

```
CALL ml_add_dnet_connection_script(  
    'ver1',  
    'end_upload',  
    'TestScripts.Test.EndUpload'  
)
```

The following is the sample .NET method EndUpload.

```
using Sap.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();
    }
}
}
}
}

```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[SQL-Java Data Types \[page 530\]](#)

[begin\\_upload Connection Event \[page 383\]](#)

[end\\_upload Table Event \[page 426\]](#)

[SQL-.NET Data Types \[page 546\]](#)

### 1.12.2.34 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2
s.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.	Not applicable

## 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.

## SQL Example

The following call to a MobiLink system procedure assigns the end\_upload event to a stored procedure called ULCustomerIDPool\_maintain.

```
CALL ml_add_table_script(
    'custdb',
    'ULCustomerIDPool',
    'end_upload',
    '{ CALL ULCustomerIDPool_maintain( {ml s.username} ) }' )
```

The following SQL statements create the ULCustomerIDPool\_maintain stored procedure. This procedure inserts new primary keys, to replace the keys used by the rows just uploaded, into a primary key pool that gets downloaded to the remote database later in the same synchronization.

```
CREATE PROCEDURE ULCustomerIDPool_maintain ( IN syncuser_id INTEGER )
BEGIN
```

```

DECLARE pool_count INTEGER;
-- Determine how many ids to add to the pool
SELECT COUNT(*) INTO pool_count
  FROM ULCustomerIDPool WHERE pool_emp_id = syncuser_id;
-- Top up the pool with new ids
WHILE pool_count < 20 LOOP
  INSERT INTO ULCustomerIDPool ( pool_emp_id ) VALUES ( syncuser_id );
  SET pool_count = pool_count + 1;
END LOOP;
END

```

## Java Example

The following call to a MobiLink system procedure registers a Java method called endUploadTable as the script for the end\_upload table event when synchronizing the script version ver1.

```

CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'end_upload',
  'ExamplePackage.ExampleClass.endUploadTable' )

```

The following is the sample Java method endUploadTable. It generates a delete for a table with a name related to the passed-in table name. This syntax is for SQL Anywhere consolidated databases.

```

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.

```

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.

```

using Sap.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();
}
}
}

```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[SQL-Java Data Types \[page 530\]](#)

[begin\\_upload Table Event \[page 385\]](#)

[end\\_upload Connection Event \[page 424\]](#)

[SQL-.NET Data Types \[page 546\]](#)

### 1.12.2.35 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2
s.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.	Not applicable

## 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.

## SQL Example

You can use this event to process rows deleted during the upload on an intermediate table. You can compare the rows in the base table with rows in the intermediate table and decide what to do with the deleted row.

The following call to a MobiLink system procedure assigns the EndUploadDeletesLeads stored procedure to the end\_upload\_deletes event.

```
CALL ml_add_table_script(
  'version1',
  'Leads',
  'end_upload_deletes',
  'call EndUploadDeletesLeads()');
```

The following SQL statement creates the EndUploadDeletes stored procedure.

```
CREATE PROCEDURE EndUploadDeletesLeads ( )
Begin
  FOR names AS curs CURSOR FOR
```

```

SELECT LeadID
  FROM Leads
 WHERE LeadID NOT IN (SELECT LeadID FROM T_Leads)
DO
  CALL decide_what_to_do( LeadID )
END FOR;
end

```

## Java Example

The following call to a MobiLink system procedure registers a Java method called endUploadDeletes as the script for the end\_upload\_deletes table event when synchronizing the script version ver1.

```

CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'end_upload_deletes',
  'ExamplePackage.ExampleClass.endUploadDeletes' )

```

The following is the sample Java method 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 table1.

```

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 );
}}}

```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[SQL-Java Data Types \[page 530\]](#)

[begin\\_upload\\_deletes Table Event \[page 388\]](#)

[SQL-.NET Data Types \[page 546\]](#)

### 1.12.2.36 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2
s.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.	Not applicable



## 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.

## SQL Example

The following call to a MobiLink system procedure registers a SQL method called EndUploadRows as the script for the end\_upload\_rows table event when synchronizing the script version ver1.

```
CALL ml_add_table_script(  
  'version1',  
  'table1',  
  'end_upload_rows',  
  'CALL EndUploadRows(  
    { ml s.username },  
    { ml s.table } )' )
```

The following is the sample SQL method EndUploadRows. It calls a SQL method that manipulates the database.

```
CREATE PROCEDURE EndUploadRows (  
  IN 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.

```
CALL ml_add_java_table_script(  
  'ver1',  
  'table1',  
  'end_upload_rows',  
  'ExamplePackage.ExampleClass.endUploadRows' )
```

The following is the sample Java method `endUploadRows`. It calls a Java method that manipulates the database.

```
public 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`.

```
CALL ml_add_dnet_table_script(
    'ver1',
    'table1',
    'end_upload_rows',
    'TestScripts.Test.EndUploadRows'
)
```

The following is the sample .NET method `endUploadRows`. It calls a .NET method that manipulates the database.

```
public void EndUploadRows (
    string user,
    string table ) {
    processUploadedRows( _syncConn, table );
}}
```

## Related Information

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[SQL-Java Data Types \[page 530\]](#)

[begin\\_upload\\_rows Table Event \[page 391\]](#)

[SQL-.NET Data Types \[page 546\]](#)

## 1.12.2.37 generate\_next\_last\_download\_timestamp Connection 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.next_last_download	TIMESTAMP. This is an INOUT parameter. The MobiLink server initializes this parameter with the last_download_timestamp, a timestamp used to generate a download stream in the current synchronization.	1
s.username	VARCHAR(128). The MobiLink user name.	2
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.	Not applicable

### 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, Microsoft Azure, 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, Microsoft SQL Server, and Microsoft Azure when the snapshot isolation level is enabled on the database.

### i Note

Support for IBM DB2 consolidated databases is deprecated.

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 timestamp-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.

## 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 using SQL:

1. Assume you have a sync table called `my_table` that is defined as follows:

```
CREATE TABLE my_table ( pk          INT PRIMARY KEY NOT NULL,
                        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:

```
CREATE PROCEDURE GenerateNextDownloadTimestamp ( p_ts IN OUT TIMESTAMP ) AS
BEGIN
  SELECT SYS_EXTRACT_UTC( NVL( MIN( TO_TIMESTAMP( START_TIME, 'mm/dd/rr
hh24:mi:ss' ) ),
                                SYSTIMESTAMP ) )
    INTO p_ts FROM GV$TRANSACTION;
END;
```

3. Call the `ml_add_connect_script` to install the script:

```
call ml_add_connection_script(
  'my_script_version',
  'generate_next_last_download_timestamp',
  '{ call GenerateNextDownloadTimestamp( {ml s.next_last_download} ) }' )
```

### i Note

The MobiLink server logon ID must have a SELECT privilege on GV\_\$TRANSACTION.

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[Last Download Times in Scripts \[page 117\]](#)

[How Download Timestamps Are Generated and Used \[page 118\]](#)

[SQL-Java Data Types \[page 530\]](#)

[modify\\_next\\_last\\_download\\_timestamp Connection Event \[page 460\]](#)

[modify\\_last\\_download\\_timestamp Connection Event \[page 457\]](#)

[SQL-.NET Data Types \[page 546\]](#)

### 1.12.2.38 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.

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.

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.

## Java Example

The following call to a MobiLink system procedure registers a Java method called `handleDownload` for the `handle_DownloadData` connection event when synchronizing the script version `ver1`. You run this system procedure against your MobiLink consolidated database.

```
CALL ml_add_java_connection_script(
  'ver1',
  'handle_DownloadData',
  'MyPackage.MobiLinkOrders.handleDownload' )
```

The following example shows you how to use the `handleDownload` method to create a download.

The following code sets up a class level `DBConnectionContext` instance in the constructor for a class called `MobiLinkOrders`.

```
import com.sap.ml.script.*;
import java.io.*;
import java.sql.*;
import java.lang.System;
public class MobiLinkOrders{
  DBConnectionContext _cc;
  public MobiLinkOrders( DBConnectionContext cc ) {
    _cc = cc;
  }
}
```

In your `HandleDownload` method, you use the `DBConnectionContext` `getDownloadData` method to return a `DownloadData` instance for the current synchronization. The `DownloadData` `getDownloadTableByName` method returns a `DownloadTableData` instance for the `remoteOrders` table. The `DownloadTableData`

getUpsertPreparedStatement method returns a java.sql.PreparedStatement. To add an operation to the download, you set all column values and call the executeUpdate method.

The following is the handleDownload method of the MobiLinkOrders class. It adds two rows to the download for a table called remoteOrders.

```
// Method used for the handle_DownloadData event.
public void handleDownload() throws SQLException {
    // Get DownloadData instance for current synchronization.
    DownloadData downloadData = _cc.getDownloadData();

    // Get a DownloadTableData instance for the remoteOrders table.
    DownloadTableData td = downloadData.getDownloadTableByName("remoteOrders");
    // Get a java.sql.PreparedStatement for upsert (update/insert) operations.
    PreparedStatement upsertPS = td.getUpsertPreparedStatement();
    // Set values for one row.
    upsertPS.setInt( 1, 2300 );
    upsertPS.setInt( 2, 100 );
    // Add the values to the download.
    int updateResult = upsertPS.executeUpdate();
    // Set values for another row.
    upsertPS.setInt( 1, 2301 );
    upsertPS.setInt( 2, 50 );
    updateResult = upsertPS.executeUpdate();
    // ...
    upsertPS.close();
}
```

## .NET Example

The following call to a MobiLink system procedure registers a .NET method called HandleDownload as the script for the handle\_DownloadData connection event when synchronizing the script version ver1. This syntax is for SQL Anywhere consolidated databases.

```
CALL ml_add_dnet_connection_script(
    'ver1', 'handle_DownloadData',
    'TestScripts.MobiLinkOrders.HandleDownload'
)
```

The following is the sample .NET method HandleDownload:

```
using System;
using System.Data;
using System.IO;
using Sap.MobiLink.Script;
using Sap.MobiLink;
namespace MyScripts
{
    /// <summary>
    /// Tests that scripts are called correctly for most sync events.
    /// </summary>
    public class MobiLinkOrders
    {
        private DBConnectionContext _cc;
        public MobiLinkOrders( DBConnectionContext cc )
        {
            _cc = cc;
        }
        ~MobiLinkOrders()
        {
        }
    }
}
```

```

}
public void handleDownload()
{
    // Get DownloadData instance for current synchronization.
    DownloadData my_dd = _cc.GetDownloadData();

    // Get a DownloadTableData instance for the remoteOrders table.
    DownloadTableData td = my_dd.GetDownloadTableByName("remoteOrders");
    // Get an IDbCommand for upsert (update/insert) operations.
    IDbCommand upsert_stmt = td.GetUpsertCommand();
    IDataParameterCollection parameters = upsert_stmt.Parameters;
    // Set values for one row.
    parameters[ 0 ] = 2300;
    parameters[ 1 ] = 100;
    // Add the values to the download.
    int update_result = upsert_stmt.ExecuteNonQuery();
    // Set values for another row.
    parameters[ 0 ] = 2301;
    parameters[ 1 ] = 50;
    update_result = upsert_stmt.ExecuteNonQuery();
    // ...
}
}
}

```

## Related Information

[Data Scripts \[page 346\]](#)

[Direct Row Handling \[page 558\]](#)

[Direct Downloads \[page 568\]](#)

[Scripts Required for Synchronization \[page 315\]](#)

[Script Additions and Deletions \[page 316\]](#)

[handle\\_UploadData Connection Event \[page 448\]](#)

[ml\\_add\\_java\\_connection\\_script System Procedure \[page 593\]](#)

### 1.12.2.39 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.action_code	INTEGER. This is an INOUT parameter. Set this value to tell MobiLink server how to respond to the error.	1
s.error_code	INTEGER. The native RDBMS error code.	2
s.error_message	TEXT. The native RDBMS error message.	3
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	4
s.table	VARCHAR(128). The table whose script had the error. If the script is not a table script, the table name is null.	5
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.	Not applicable

## Default Action

The MobiLink server selects a default action. You can modify the action in the script, and return a value instructing MobiLink how to proceed. The `action_code` parameter takes one of the following values:

### 1000

Skip the current row and continue processing.

### 3000

Rollback the current transaction and cancel the current synchronization. This is the default action code, and is used when no `handle_error` script is defined or this script causes an error.

### 4000

Rollback the current transaction, cancel the synchronization, and shut down the MobiLink server.

## Remarks

The MobiLink server sends in the current action code. Initially, this is set to 3000 for each set of errors caused by a single SQL operation. Usually, there is only one error per SQL operation, but there may be more. If

uploading rows in batches using the `-s mlsrv17` option, 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.

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 on the severity of the error. Your script may modify this value. Your script must return or set an action code.

The `error_code` and `message` allow you to identify the nature of the error.

The MobiLink server executes this script if an ODBC error occurs while MobiLink is processing an insert, update, or delete script during the upload transaction or is fetching download rows. If an ODBC error occurs at another time, the MobiLink server calls the `report_error` or `report_odbc_error` script and aborts the synchronization.

If the error happened while manipulating a particular table, the table name is supplied. Otherwise, this value is null. The table name is the name of a table in the client application. This name may or may not have a direct counterpart in the consolidated database, depending upon how your remote table names map to consolidated tables.

SQL scripts for the `handle_error` event must be implemented as stored procedures.

You can return a value from the `handle_error` script one of the following ways:

- Pass the `action_code` parameter to an `OUTPUT` parameter of a procedure:

```
CALL my_handle_error( {ml s.action_code}, {ml s.error_code}, {ml
s.error_message}, {ml s.username}, {ml s.table} )
```

- Set the `action_code` via a procedure or function return value:

```
{ml s.action_code} = CALL my_handle_error( {ml s.error_code}, {ml
s.error_message}, {ml s.username}, {ml s.table} )
```

Most RDBMSs use the `RETURN` statement to set the return value from a procedure or function.

The CustDB sample application contains error handlers for various database-management systems.

## 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.

```
package ExamplePackage;
public class ExampleClass
{
  public void handleError( com.sap.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.

```
CALL ml_add_dnet_connection_script(  
    'ver1',  
    'handle_error',  
    'TestScripts.Test.HandleError' )
```

The following is the sample .NET method HandleError.

```
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;  
            }  
        }  
    }  
}
```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[Data Scripts \[page 346\]](#)

[SQL-Java Data Types \[page 530\]](#)

[report\\_error Connection Event \[page 476\]](#)

[report\\_odbc\\_error Connection Event \[page 480\]](#)

[handle\\_odbc\\_error Connection Event \[page 445\]](#)

[-s mlsrv17 Option \[page 81\]](#)

[SQL-.NET Data Types \[page 546\]](#)

## 1.12.2.40 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.action_code	INTEGER. This is an INOUT parameter. Set this value to tell the MobiLink server how to respond to the error.	1
s.odbc_state	VARCHAR(5). The ODBC SQLSTATE.	2
s.error_message	TEXT. The ODBC error message	3
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	4
s.table	VARCHAR(128). The table name.	5
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.	Not applicable

### Default Action

The MobiLink server selects a default action. You can modify the action in the script, and return a value instructing MobiLink how to proceed. The action\_code parameter takes one of the following values:

#### 1000

Skip the current row and continue processing.

#### 3000

Rollback the current transaction and cancel the current synchronization. This is the default action code, and is used when no handle\_error script is defined or this script causes an error.

## 4000

Rollback the current transaction, cancel the synchronization, and shut down the MobiLink server.

## Remarks

The MobiLink server executes this script whenever it encounters an error flagged by the ODBC Driver Manager if the error occurs while MobiLink is processing an insert, update, or delete script during the upload transaction or is fetching download rows. If an ODBC error occurs at another time, the MobiLink server calls the `report_error` or `report_odbc_error` script and aborts the synchronization.

The error codes allow you to identify the nature of the error.

The action code tells the MobiLink server what to do next. Before it calls this script, the MobiLink server sets the action code to a default value, which depends on 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.

## SQL Example

The following example works with a SQL Anywhere consolidated database. It allows your application to ignore ODBC integrity constraint violations.

The following call to a MobiLink system procedure assigns the `HandleODBCError` stored procedure to the `handle_odbc_error` event.

```
CALL ml_add_connection_script(
  'ver1',
  'handle_odbc_error',
  'CALL HandleODBCError(
    {ml s.action_code},
    {ml s.ODBC_state},
    {ml s.error_message},
    {ml s.username},
    {ml s.table} )' )
```

The following SQL statement creates the `HandleODBCError` stored procedure.

```
CREATE PROCEDURE HandleODBCError(
  INOUT action integer,
  IN odbc_state varchar(5),
  IN error_message varchar(1000),
  IN user_name varchar(128),
  IN table_name varchar(128) )
BEGIN
  IF odbc_state = '23000' then
```

```

-- Ignore the error and keep going.
SET action = 1000;
ELSE
-- Abort the synchronization.
SET action = 3000;
END IF;
END

```

## Java Example

The following call to a MobiLink system procedure registers a Java method called `handleODBCError` as the script for the `handle_odbc_error` event when synchronizing the script version `ver1`.

```

CALL ml_add_java_connection_script(
'ver1',
'handle_odbc_error',
'ExamplePackage.ExampleClass.handleODBCError'
)

```

The following is the sample Java method `handleODBCError`. It processes an error based on the data that is passed in. It also determines the resulting error code.

```

package ExamplePackage;
public class ExampleClass
{
    public void handleODBCError( com.sap.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;
            }
        }
    }
}

```

## Related Information

- [Script Parameters \[page 294\]](#)
- [Script Additions and Deletions \[page 316\]](#)
- [Remote IDs and MobiLink User Names in Scripts](#)
- [SQL-Java Data Types \[page 530\]](#)
- [handle\\_error Connection Event \[page 440\]](#)
- [report\\_error Connection Event \[page 476\]](#)
- [report\\_odbc\\_error Connection Event \[page 480\]](#)
- [SQL-.NET Data Types \[page 546\]](#)

### 1.12.2.41 handle\_UploadData Connection Event

A non-SQL data script used by direct row handling to process uploaded rows.

#### Parameters

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
UploadData	A .NET or Java class encapsulating table operations uploaded by a MobiLink client. This class is defined in the MobiLink server API for Java and .NET.	1



Parameter name for SQL scripts	Description	Order (deprecated for SQL)
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.	Not applicable

## 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.

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.

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.

### **i** Note

This event cannot be implemented as SQL.

## Java Examples

The following call to a MobiLink system procedure registers a Java method called `handleUpload` for the `handle_UploadData` connection event when synchronizing the script version `ver1`. You run this system procedure against your MobiLink consolidated database.

```
CALL ml_add_java_connection_script(
    'ver1',
    'handle_UploadData',
    'MyPackage.MyClass.handleUpload' )
```

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.

```

package MyPackage;
import com.sap.ml.script.*;
import java.sql.*;
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 com.sap.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.

```
public void printRSInfo( ResultSet results )
    throws SQLException, IOException {

    // Obtain the result set metadata.
    ResultSetMetaData metaData = results.getMetaData();
    String columnHeading = "";
    // Print out column headings.
    for( int c = 1; c <= metaData.getColumnCount(); c++ ) {
        columnHeading += metaData.getColumnLabel(c);
        if( c < metaData.getColumnCount() ) {
            columnHeading += ", ";
        }
    }
    println( columnHeading );
    while( results.next() ) {
        // Print out each row.
        printRow( results, metaData.getColumnCount() );
    }
    // Close the java.sql.ResultSet.
    results.close();
}
```

The `printRow` method, shown below, uses the `ResultSet` `getString` method to obtain each column value.

```
public void printRow( ResultSet results, int colCount )
    throws SQLException, IOException {
    String row = "( ";

    for( int c = 1; c <= colCount; c++ ) {
        // Get a column value.
        String currentColumn = results.getString( c );

        // Check for null values.
        if( currentColumn == null ) {
            currentColumn = "<NULL>";
        }
        // Add the column value to the row string.
        row += cur_col;
        if( c < colCount ) {
            row += ", ";
        }
    }
    row += " )";
    // Print out the row.
    println( row );
}
```

## .NET Example

The following call to a MobiLink system procedure registers a .NET method called `HandleUpload` for the `handle_UploadData` connection event when synchronizing the script version `ver1`. You run this system procedure against your MobiLink consolidated database.

```
CALL ml_add_dnet_connection_script(
    'ver1',
    'handle_UploadData',
```

```
'TestScripts.Test.HandleUpload' )
```

The following .NET method processes the upload for the remoteOrders table. This example makes use of the SetOldRowValues and SetNewRowValues methods to access both the pre-image and post-image of each update.

```
using System;
using System.Data;
using System.IO;
using Sap.MobiLink.Script;
using Sap.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" );
            printUpdaterRSInfo( cur_table.GetUpdates() );

            // Print out delete result set.
            Console.Write( "Deletes" );
            printRSInfo( cur_table.GetDeletes() );
        }
    public void printRSInfo( IDataReader dr )
    {
        // Obtain the result set metadata.
        DataTable dt = dr.GetSchemaTable();
        DataColumnCollection cc = dt.Columns;
        DataColumn dc;
        String columnHeading = "";
        // Print out column headings.
        for( int c=0; c < cc.Count; c = c + 1 ) {
            dc = cc[ c ];
            columnHeading += dc.ColumnName;
            if( c < cc.Count - 1 ) {
                columnHeading += ", ";
            }
        }
        Console.Write( columnHeading );
        while( dr.Read() ) {
            // Print out each row.
            printRow( dr, cc.Count );
        }
        // Close the java.sql.ResultSet.
        dr.Close();
    }
    public void printUpdaterRSInfo( UpdateDataReader utr )
    {
        // Obtain the result set metadata.
```

```

        DataTable dt = utr.GetSchemaTable();
        DataColumnCollection cc = dt.Columns;
        DataColumn dc;
        String columnHeading = "TYPE, ";
        // Print out column headings.
        for( int c = 0; c < cc.Count; c = c + 1 ) {
            dc = cc[ c ];
            columnHeading += dc.ColumnName;
            if( c < cc.Count - 1 ) {
                columnHeading += ", ";
            }
        }
        Console.Write( columnHeading );
        while( utr.Read() ) {
            // Print out the new values for the row.
            utr.SetNewRowValues();
            Console.Write( "NEW:" );
            printRow( utr, cc.Count );
            // Print out the old values for the row.
            utr.SetOldRowValues();
            Console.Write( "OLD:" );
            printRow( utr, cc.Count );
        }
        // Close the java.sql.ResultSet.
        utr.Close();
    }
    public void printRow( IDataReader dr, int col_count )
    {
        String row = "( ";
        int c;

        for( c = 0; c < col_count; c = c + 1 ) {
            // Get a column value.
            String cur_col = dr.GetString( c );

            // Check for null values.
            if( cur_col == null ) {
                cur_col = "<NULL>";
            }
            // Add the column value to the row string.
            row += cur_col;
            if( c < col_count ) {
                row += ", ";
            }
        }
        row += " )";
        // Print out the row.
        Console.Write( row );
    }
}

```

## Related Information

[Data Scripts \[page 346\]](#)

[Direct Row Handling \[page 558\]](#)

[Direct Uploads \[page 563\]](#)

[Direct Upload Conflicts \[page 564\]](#)

[Scripts Required for Synchronization \[page 315\]](#)

[Script Additions and Deletions \[page 316\]](#)

[handle\\_DownloadData Connection Event \[page 437\]](#)

[ml\\_add\\_java\\_connection\\_script System Procedure \[page 593\]](#)

## 1.12.2.42 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.error_message	VARBINARY(1024). This is an INOUT parameter, representing the error message.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
s.error_code	INTEGER. The MobiLink error code associated with the error_message.	3
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.	Not applicable

### 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.

## SQL Example

The following example downloads everything from one day ago, regardless of whether the databases were synchronized since then.

The following SQL statement creates the ModifyLastErrorMessage stored procedure:

```
CREATE PROCEDURE ModifyLastErrorMessage(
  inout error_message VARBINARY(1024),
  in username VARCHAR(128),
  in error_code INT )
BEGIN
  SELECT dateadd(day, -1, last_download_time )
  INTO last_download_time
END
```

The following call to a MobiLink system procedure assigns ModifyLastErrorMessage to the modify\_error\_message connection event for the script version modify\_ts\_test:

```
CALL ml_add_connection_script(
  'modify_ts_test',
  'modify_error_message',
  'CALL ModifyLastErrorMessage (
    {ml s.error_message},
    {ml s.username},
    {ml s.error_code} )' );
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called modifyLastErrorMessage as the script for the modify\_error\_message connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
  'ver1',
  'modify_error_message',
  'ExamplePackage.ExampleClass.modifyLastErrorMessage' )
```

The following is the sample Java method modifyLastErrorMessage. It prints the current error message and error code.

```
package ExamplePackage;
public class ExampleClass {
  String _curUser = null;
  public void modifyLastErrorMessage(
    com.sap.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`.

```
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.

```
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 );
}}}
```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[SQL-Java Data Types \[page 530\]](#)

[SQL-.NET Data Types \[page 546\]](#)



## 1.12.2.43 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.last_download	TIMESTAMP. The oldest download time for any synchronized table. This is an INOUT parameter.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
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.	Not applicable

### 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.

## 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:

```
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 Microsoft SQL Server consolidated databases:

```
CREATE PROCEDURE ModifyDownloadTimestamp
    @download_timestamp DATETIME OUTPUT,
    @user_name          VARCHAR( 128 )
AS
BEGIN
    -- N is the maximum replication latency in consolidated cluster
    SELECT @download_timestamp = @download_timestamp - N
END
```

The following call to a MobiLink system procedure assigns the `ModifyDownloadTimestamp` stored procedure to the `modify_last_download_timestamp` event. The following syntax is for a SQL Anywhere consolidated database:

```
CALL ml_add_connection_script(
    'my_version',
    'modify_last_download_timestamp',
    '{CALL ModifyDownloadTimestamp(
    {ml s.last_download},
    {ml s.username} ) }' )
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called `modifyLastDownloadTimestamp` as the script for the `modify_last_download_timestamp` connection event when synchronizing the script version `ver1`.

```
CALL ml_add_java_connection_script(
  'ver1',
  'modify_last_download_timestamp',
  'ExamplePackage.ExampleClass.modifyLastDownloadTimestamp' )
```

The following is the sample Java method `modifyLastDownloadTimestamp`. It prints the current and new timestamp and modifies the timestamp that is passed in.

```
public 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`.

```
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() );
}
```

## Related Information

[Script Parameters \[page 294\]](#)

- [Script Additions and Deletions \[page 316\]](#)
- [Remote IDs and MobiLink User Names in Scripts](#)
- [Last Download Times in Scripts \[page 117\]](#)
- [How Download Timestamps Are Generated and Used \[page 118\]](#)
- [SQL-Java Data Types \[page 530\]](#)
- [modify\\_next\\_last\\_download\\_timestamp Connection Event \[page 460\]](#)
- [generate\\_next\\_last\\_download\\_timestamp Connection Event \[page 435\]](#)
- [SQL-.NET Data Types \[page 546\]](#)

## 1.12.2.44 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.next_last_download	TIMESTAMP. This is an INOUT parameter. The MobiLink server generates this value immediately after the upload is committed.	1
s.last_download	TIMESTAMP. This is the last download time for the current synchronization.	2
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	3

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
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.	Not applicable

## 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.

## SQL Example

The following example shows one application of this script. Create a stored procedure. The following syntax is for a SQL Anywhere consolidated database:

```
CREATE PROCEDURE ModifyNextDownloadTimestamp(
  inout next_last_download TIMESTAMP ,
  in last_download TIMESTAMP ,
  in user_name VARCHAR(128) )
BEGIN
  SELECT dateadd(hour, -1, next_last_download )
  INTO next_last_download
END
```

Install the script into your SQL Anywhere consolidated database:

```
CALL ml_add_connection_script(
  'modify_ts_test',
  'modify_next_last_download_timestamp',
```

```
'CALL ModifyNextDownloadTimestamp (
  {ml s.next_last_download},
  {ml s.last_download},
  {ml s.username} )' )
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called `modifyNextDownloadTimestamp` as the script for the `modify_next_last_download_timestamp` connection event when synchronizing the script version `ver1`.

```
CALL ml_add_java_connection_script(
  'ver1',
  'modify_next_last_download_timestamp',
  'ExamplePackage.ExampleClass.modifyNextDownloadTimestamp' )
```

The following is the sample Java method `modifyNextDownloadTimestamp`. It sets the download timestamp back an hour.

```
public 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.

```
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 );
  }}
}}
```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[Last Download Times in Scripts \[page 117\]](#)

[How Download Timestamps Are Generated and Used \[page 118\]](#)

[SQL-Java Data Types \[page 530\]](#)

[modify\\_last\\_download\\_timestamp Connection Event \[page 457\]](#)

[generate\\_next\\_last\\_download\\_timestamp Connection Event \[page 435\]](#)

[SQL-.NET Data Types \[page 546\]](#)

## 1.12.2.45 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name. This is an INOUT parameter.	1
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.	Not applicable

## 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.

### **i** Note

A more flexible approach to mapping the MobiLink user name is to use user-defined named parameters.

## 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.

```
package ExamplePackage;  
import java.lang.Integer;  
import java.sql.*;  
import com.sap.ml.script.*;  
public 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.

```
CALL ml_add_dnet_connection_script(  
    'ver1',  
    'modify_user',  
    'TestScripts.Test.ModUser'  
)
```

The following is the sample .NET method ModUser.

```
using Sap.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];
            }
        }
    }
}
```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[User-defined Named Parameters \[page 310\]](#)

[SQL-Java Data Types \[page 530\]](#)

[authenticate\\_user Connection Event \[page 354\]](#)

[authenticate\\_user\\_hashed Connection Event \[page 360\]](#)

[SQL-.NET Data Types \[page 546\]](#)

### 1.12.2.46 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.last_download	TIMESTAMP. This is the last download time for the current synchronization.	2
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.	Not applicable

## 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.

### **i** 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 com.sap.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.

```
using System;
using Sap.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();
        }
    }
}
```

## Related Information

[SQL-Java Data Types \[page 530\]](#)

[publication\\_nonblocking\\_download\\_ack Connection Event \[page 472\]](#)

[SendDownloadAck \(sa\) Extended Option](#)

[Send Download Acknowledgement Synchronization Parameter](#)

[SQL-.NET Data Types \[page 546\]](#)

## 1.12.2.47 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.last_download	TIMESTAMP. The oldest download time of any synchronized table.	1
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	2
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.	Not applicable

### 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.

## SQL Example

The following call to a MobiLink system procedure registers a SQL stored procedure called `prepareForDownload` as the script for the `prepare_for_download` event when synchronizing the script version `ver1`.

```
CALL ml_add_connection_script(  
    'ver1',  
    'prepare_for_download',  
    'CALL prepareForDownload(  
        { ml s.username } )' )
```

The following is the sample SQL method `prepareForDownload`. This stored procedure prepares downloads for two tables. For example, it could take information from many tables and store it in temporary tables referenced by the `download_cursor` scripts for tables T1 and T2.

```
CREATE PROCEDURE prepareForDownload (  
    IN ts TIMESTAMP,  
    IN "user" VARCHAR(128))  
BEGIN  
    CALL prepareT1Download( user, ts );  
    CALL prepareT2Download( user, ts );  
END;
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called `prepareForDownload` as the script for the `prepare_for_download` event when synchronizing the script version `ver1`.

```
CALL ml_add_java_connection_script(  
    'ver1',  
    'prepare_for_download',  
    'ExamplePackage.ExampleClass.prepareForDownload' )
```

The following is the sample Java method `prepareForDownload`. This method prepares downloads for two tables. For example, it could take information from many tables, plus other information accessible from Java, and store it in temporary tables referenced by the `download_cursor` scripts for tables T1 and T2.

```
public void prepareForDownload(  
    Timestamp ts,  
    String user ) {  
    prepareT1ForDownload( _synconn, user, ts );  
    prepareT2ForDownload( _synconn, 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`.

```
CALL ml_add_dnet_connection_script(  
    'ver1',  
    'prepare_for_download',  
    'ExamplePackage.ExampleClass.PrepareForDownload' )
```

```
'ver1',
'prepare_for_download',
'TestScripts.Test.PrepareForDownload'
)
```

The following is the sample .NET method PrepareForDownload. This method prepares downloads for two tables. For example, it could take information from many tables, plus other information accessible from .NET, and store it in temporary tables referenced by the download\_cursor scripts for tables T1 and T2.

```
public void PrepareForDownload(
    DateTime ts,
    string user ) {
    PrepareT1ForDownload ( _syncConn, user, ts );
    PrepareT2ForDownload ( _syncConn, user, ts );
}
```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[Last Download Times in Scripts \[page 117\]](#)

[SQL-Java Data Types \[page 530\]](#)

[end\\_upload Connection Event \[page 424\]](#)

[begin\\_download Connection Event \[page 366\]](#)

[SQL-.NET Data Types \[page 546\]](#)

### 1.12.2.48 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.last_publication_download	TIMESTAMP. The earliest last download time of any synchronized table.	2
s.publication name	VARCHAR(128). The name of the publication.	3
s.subscription_id	VARCHAR(128). The remote subscription ID.	4
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.	Not applicable

## 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.

### **i** 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.

## SQL Example

The following script adds a record to a table called `download_pubs_acked`. The record contains the publication name, the first authentication parameter, and a download timestamp.

```
INSERT INTO download_pubs_acked( pub_name, auth_parm, last_download )
VALUES( {ml s.publication_name}, {ml a.1}, {ml s.last_publication_download} )
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called `publicationDownloadACK` as the script for the `publication_nonblocking_download_ack` connection event when synchronizing the script version `ver1`.

```
CALL ml_add_java_connection_script (
  'ver1',
  'publication_nonblocking_download_ack',
  'ExamplePackage.ExampleClass.publicationDownloadACK' )
```

The following is the sample Java method `publicationDownloadACK`. It performs some business logic by acting on the confirmation if a particularly important publication was downloaded.

```
package ExamplePackage;
import com.sap.ml.script.*;
import java.sql.*;
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 Sap.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;  
            cmd.Parameters[2] = pubName;  
            cmd.ExecuteNonQuery();  
        }  
    }  
}
```

## Related Information

[SQL-Java Data Types \[page 530\]](#)

[nonblocking\\_download\\_ack Connection Event \[page 466\]](#)

[SendDownloadAck \(sa\) Extended Option](#)

[Send Download Acknowledgement Synchronization Parameter](#)

[SQL-.NET Data Types \[page 546\]](#)

## 1.12.2.49 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.action_code	INTEGER. This is an INOUT parameter. This parameter is mandatory.	1
s.error_code	INTEGER. The native DBMS error code.	2
s.error_message	TEXT. The native DBMS error message.	3
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	4
s.table	VARCHAR(128). The table whose script caused the error.	5
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.	Not applicable

### 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 `handle_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.

## SQL Example

The following example works with a SQL Anywhere consolidated database. It inserts a row into a table used to record synchronization errors.

```
CALL ml_add_connection_script(
  'ver1',
  'report_error',
  'INSERT INTO sync_error(
    action_code,
    error_code,
    error_message,
    user_name,
    table_name )
VALUES (
  {ml s.action_code},
  {ml s.error_code},
  {ml s.error_message},
  {ml s.username},
  {ml s.table} )' )
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called reportError as the script for the report\_error connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(  
    'ver1',  
    'report_error',  
    'ExamplePackage.ExampleClass.reportError' )
```

The following is the sample Java method reportError. It logs the error to a table using the JDBC connection provided by MobiLink. It also sets the action code.

```
package ExamplePackage;  
import java.sql.*;  
import com.sap.ml.script.*;  
public class ExampleClass  
{  
    DBConnectionContext _cc;  
  
    public ExampleClass( DBConnectionContext cc )  
    {  
        _cc = cc;  
    }  
    public void reportError( com.sap.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.

```
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.

```
using System;
using Sap.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[1] = errorCode;
            cmd.Parameters[2] = errorMessage;
            cmd.Parameters[3] = user;
            cmd.Parameters[4] = table;
            cmd.ExecuteNonQuery();
        }
    }
}
```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[SQL-Java Data Types \[page 530\]](#)

[handle\\_error Connection Event \[page 440\]](#)

[handle\\_odbc\\_error Connection Event \[page 445\]](#)

[report\\_odbc\\_error Connection Event \[page 480\]](#)

[SQL-.NET Data Types \[page 546\]](#)

## 1.12.2.50 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.action_code	INTEGER. This is an INOUT parameter. This parameter is mandatory.	1
s.odbc_state	VARCHAR(5). The ODBC SQLSTATE.	2
s.error_message	TEXT. The ODBC error message.	3
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	4
s.table	VARCHAR(128). The table whose script caused the error.	5
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.	Not applicable

### 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.

## SQL Example

The following example works with a SQL Anywhere consolidated database. It inserts a row into a table used to record synchronization errors.

```
CALL ml_add_connection_script(
  'ver1',
  'report_odbc_error',
  'INSERT INTO sync_error(
    action_code,
    odbc_state,
    error_message,
    user_name,
    table_name )
  VALUES (
    {ml s.action_code},
    {ml s.odbc_state},
    {ml s.error_message},
    {ml s.username},
    {ml s.table} )' )
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called `reportODBCError` as the script for the `report_odbc_error` event when synchronizing the script version `ver1`.

```
CALL ml_add_java_connection_script(
  'ver1',
  'report_odbc_error',
  'ExamplePackage.ExampleClass.reportODBCError' )
```

The following is the sample Java method reportODBCError. It logs the error to a table using the JDBC connection provided by MobiLink. It also sets the action code.

```
package ExamplePackage;
import java.sql.*;
import com.sap.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 Sap.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[2] = errorMessage;
        cmd.Parameters[3] = user;
        cmd.Parameters[4] = table;
        cmd.ExecuteNonQuery();
    }
}
}

```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[SQL-Java Data Types \[page 530\]](#)

[handle\\_error Connection Event \[page 440\]](#)

[handle\\_odbc\\_error Connection Event \[page 445\]](#)

[report\\_error Connection Event \[page 476\]](#)

[SQL-.NET Data Types \[page 546\]](#)

### 1.12.2.51 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2
s.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.	Not applicable

## 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.

## SQL Example

The following statement defines a `resolve_conflict` script suited to the CustDB sample application for an Oracle installation. It calls a stored procedure `ULResolveOrderConflict`.

```
exec ml_add_table_script(
  'custdb', 'ULOrder', 'resolve_conflict',
  'begin ULResolveOrderConflict();
end; ')
CREATE OR REPLACE PROCEDURE ULResolveOrderConflict()
AS
  new_order_id integer;
  new_status   varchar(20);
  new_notes   varchar(50);
BEGIN
  -- approval overrides denial
  SELECT order_id, status, notes
    INTO new_order_id, new_status, new_notes
    FROM ULNewOrder
   WHERE syncuser_id = SyncUserID;
  IF new_status = 'Approved' THEN
    UPDATE ULOrder o
      SET o.status = new_status, o.notes =
        new_notes
     WHERE o.order_id = new_order_id;
  END IF;
  DELETE FROM ULOldOrder;
  DELETE FROM ULNewOrder;
END;
```

## Java Example

The following call to a MobiLink system procedure registers a Java method called `resolveConflict` as the script for the `resolve_conflict` table event when synchronizing the script version `ver1`.

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'resolve_conflict',
  'ExamplePackage.ExampleClass.resolveConflict' )
```

The following is the sample Java method `resolveConflict`. It calls a Java method that uses the JDBC connection provided by MobiLink to resolve the conflict.

```
package ExamplePackage;
import java.sql.*;
import com.sap.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.

```

using System;
using Sap.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;
                DBCommand cmd = conn.CreateCommand();
                cmd.CommandText =
                    "SELECT order_id, new_status, new_notes " +
                    "FROM " + conflictTable +
                    "WHERE rid = " + _cc.GetRemoteID();
                DBRowReader dr = cmd.ExecuteReader();
            }
        }
    }
}

```

```

        DBCommand 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();
    }
}
}
}

```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[SQL-Java Data Types \[page 530\]](#)

[upload\\_old\\_row\\_insert Table Event \[page 510\]](#)

[upload\\_new\\_row\\_insert Table Event \[page 508\]](#)

[upload\\_update Table Event \[page 522\]](#)

[end\\_upload\\_rows Table Event \[page 432\]](#)

[SQL-.NET Data Types \[page 546\]](#)

### 1.12.2.52 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.warnings	INTEGER. The number of warnings issued during the synchronization.	2
s.errors	INTEGER. The number of errors that occurred during the synchronization.	3
s.deadlocks	INTEGER. The number of deadlocks in the consolidated database that were detected for the synchronization.	4
s.synchronized_tables	INTEGER. The number of client tables that were involved in the synchronization.	5
s.connection_retries	INTEGER. The number of times the MobiLink server retried the connection to the consolidated database.	6
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.	Not applicable

## 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.

### **i** 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.



## SQL Example

The following example inserts synchronization statistics into the sync\_con\_audit table.

```
CALL ml_add_connection_script(
  'ver1',
  'synchronization_statistics',
  'INSERT INTO sync_con_audit(
    ml_user,
    warnings,
    errors,
    deadlocks,
    synchronized_tables,
    connection_retries)
  VALUES (
    {ml s.username},
    {ml s.warnings},
    {ml s.errors},
    {ml s.deadlocks},
    {ml s.synchronized_tables},
    {ml s.connection_retries})' )
```

Once statistics are inserted into the audit table, you may use these statistics to monitor your synchronizations and make optimizations where applicable.

## Java Example

The following call to a MobiLink system procedure registers a Java method called synchronizationStatisticsConnection as the script for the synchronization\_statistics connection event when synchronizing the script version ver1.

```
CALL ml_add_java_connection_script(
  'ver1',
  'synchronization_statistics',
  'ExamplePackage.ExampleClass.synchronizationStatisticsConnection'
)
```

The following is the sample Java method synchronizationStatisticsConnection. It logs some of the statistics to the MobiLink message log. (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 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.)

```
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 );  
        }  
    }  
}
```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[MobiLink Profiler \[page 247\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[SQL-Java Data Types \[page 530\]](#)

[download\\_statistics Connection Event \[page 398\]](#)

[download\\_statistics Table Event \[page 401\]](#)

[upload\\_statistics Connection Event \[page 513\]](#)

[upload\\_statistics Table Event \[page 517\]](#)

[synchronization\\_statistics Table Event \[page 491\]](#)

[time\\_statistics Connection Event \[page 494\]](#)

[time\\_statistics Table Event \[page 497\]](#)

[SQL-.NET Data Types \[page 546\]](#)

## 1.12.2.53 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2
s.warnings	INTEGER. The number of warnings that occurred for the table during the synchronization.	3
s.errors	INTEGER. The number of errors that were related to the table during the synchronization.	4
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.	Not applicable

### 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.

## SQL Example

The following example inserts synchronization statistics into the sync\_tab\_audit table.

```
CALL ml_add_table_script(
  'ver1',
  'table1',
  'upload_insert',
  'INSERT INTO sync_tab_audit (
    ml_user,
    table,
    warnings,
    errors)
  VALUES (
    {ml s.username},
    {ml s.table},
    {ml s.warnings},
    {ml s.errors} ) ' )
```

Once synchronization statistics are inserted into the audit table, you may use these statistics to monitor your synchronizations and make optimizations where applicable.

## Java Example

The following call to a MobiLink system procedure registers a Java method called synchronizationStatisticsTable as the script for the synchronization\_statistics table event when synchronizing the script version ver1.

```
CALL ml_add_java_table_script(
  'ver1',
  'table1',
  'synchronization_statistics',
  'ExamplePackage.ExampleClass.synchronizationStatisticsTable'
)
```

The following is the sample Java method synchronizationStatisticsTable. It logs some of the statistics to the MobiLink message log. (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 + " 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 );
}}}
```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[MobiLink Profiler \[page 247\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[SQL-Java Data Types \[page 530\]](#)

[download\\_statistics Connection Event \[page 398\]](#)

[download\\_statistics Table Event \[page 401\]](#)

[upload\\_statistics Connection Event \[page 513\]](#)

[upload\\_statistics Table Event \[page 517\]](#)

[synchronization\\_statistics Connection Event \[page 487\]](#)

[time\\_statistics Connection Event \[page 494\]](#)

[time\\_statistics Table Event \[page 497\]](#)

[SQL-.NET Data Types \[page 546\]](#)

## 1.12.2.54 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.event_name	VARCHAR(128). The event whose statistics are being reported.	2
s.number_of_calls	INTEGER. The number of times the script was called.	3
s.minimum_time	INTEGER. Milliseconds. The shortest time it took to execute a script during this synchronization.	4
s.maximum_time	INTEGER. Milliseconds. The longest time it took to execute a script during this synchronization.	5
s.total_time	INTEGER. Milliseconds. The total time it took to execute all scripts in the synchronization. (This is not the same as the length of the synchronization.)	6
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.	Not applicable

## 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.

## 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',
    'TestScripts.Test.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 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 );
            }
        }
    }
}

```

## Related Information

[Script Parameters \[page 294\]](#)



[Script Additions and Deletions \[page 316\]](#)  
[MobiLink Profiler \[page 247\]](#)  
[Remote IDs and MobiLink User Names in Scripts](#)  
[SQL-Java Data Types \[page 530\]](#)  
[time\\_statistics Table Event \[page 497\]](#)  
[download\\_statistics Connection Event \[page 398\]](#)  
[download\\_statistics Table Event \[page 401\]](#)  
[upload\\_statistics Connection Event \[page 513\]](#)  
[upload\\_statistics Table Event \[page 517\]](#)  
[synchronization\\_statistics Connection Event \[page 487\]](#)  
[synchronization\\_statistics Table Event \[page 491\]](#)  
[SQL-.NET Data Types \[page 546\]](#)

## 1.12.2.55 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2
s.event_name	VARCHAR(128). The event whose statistics are being reported.	3
s.number_of_calls	INTEGER. The number of times the script was called.	4

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.minimum_time	INTEGER. Milliseconds. The shortest time it took to execute a script during the synchronization of this table.	5
s.maximum_time	INTEGER. Milliseconds. The longest time it took to execute a script during the synchronization of this table.	6
s.total_time	INTEGER. Milliseconds. The total time it took to execute all scripts in the synchronization of the table. (This is not the same as the length of the synchronization.)	7
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.	Not applicable

## 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.

## SQL Example

The following example inserts statistical information into the `time_statistics` table.

```
CALL ml_add_table_script (
  'ver1',
  'table1',
  'time_statistics',
  'INSERT INTO time_statistics(
    ml_user,
    table,
    event_name,
    number_of_calls,
```

```

    minimum_time,
    maximum_time,
    total_time)
VALUES (
    {ml s.username},
    {ml s.table},
    {ml s.event_name},
    {ml s.number_of_calls},
    {ml s.minimum_time},
    {ml s.maximum_time},
    {ml s.total_time} )' );

```

## Java Example

The following call to a MobiLink system procedure registers a Java method called `timeStatisticsTable` as the script for the `time_statistics` table event when synchronizing the script version `ver1`.

```

CALL ml_add_java_table_script(
    'ver1',
    'table1',
    'time_statistics',
    'ExamplePackage.ExampleClass.timeStatisticsTable' )

```

The following is the sample Java method `timeStatisticsTable`. It prints statistics for the `upload_old_row_insert` event.

```

public 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 totTime ) {
    if( event_name == "upload_old_row_insert") {
        System.Console.WriteLine(
            "upload_old_row_insert num_calls: " + num_calls +
            "total_time: " + total_time );
    }
}
```

## Related Information

[Script Parameters \[page 294\]](#)  
[Script Additions and Deletions \[page 316\]](#)  
[MobiLink Profiler \[page 247\]](#)  
[Remote IDs and MobiLink User Names in Scripts](#)  
[SQL-Java Data Types \[page 530\]](#)  
[time\\_statistics Connection Event \[page 494\]](#)  
[download\\_statistics Connection Event \[page 398\]](#)  
[download\\_statistics Table Event \[page 401\]](#)  
[upload\\_statistics Connection Event \[page 513\]](#)  
[upload\\_statistics Table Event \[page 517\]](#)  
[synchronization\\_statistics Connection Event \[page 487\]](#)  
[synchronization\\_statistics Table Event \[page 491\]](#)  
[SQL-.NET Data Types \[page 546\]](#)

### 1.12.2.56 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. Use named parameters instead. 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	Description	Order (deprecated for SQL)
<code>s.remote_id</code>	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	N/A
<code>s.username</code>	VARCHAR(128). The MobiLink user name. This parameter is optional.	Not applicable
<code>s.script_version</code>	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.	Not applicable
<code>r.pk-column-1</code>	Required. The first deleted primary key column value, referenced by column name or column number.	1
...	...	...
<code>r.pk-column-N</code>	Required. The last deleted primary key column value, referenced by column name or column number.	N

## 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, use direct row handling.

### **i** Note

Conflict detection is usually performed much faster when done all at once in the upload\_update script.

## SQL Example

This example is taken from the Contact sample and can be found in `Samples\MobiLink>Contact\build_consol.sql`. It marks customers that are deleted from the remote database as inactive.

```
CALL ml_add_table_script(
  'ver1',
  'Customer',
  'upload_delete',
  'UPDATE Customer
   SET active = 0
   WHERE cust_id={ml r.cust_id}' )
```

## Related Information

[Direct Row Handling \[page 558\]](#)

[Data Scripts \[page 346\]](#)

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[upload\\_insert Table Event \[page 506\]](#)

[upload\\_update Table Event \[page 522\]](#)

### 1.12.2.57 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. Use named parameters instead. 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	Description	Order (deprecated for SQL)
<code>s.remote_id</code>	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
<code>s.username</code>	VARCHAR(128). The MobiLink user name. This parameter is optional.	Optional
<code>s.script_version</code>	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.	Not applicable
<code>r.primary-key-1</code>	Required. The first primary key column value, referenced by column name or column number.	1 (2 if username is referenced)
<code>r.primary-key-2</code>	Required. The second primary key column value, referenced by column name or column number.	2
...	...	...
<code>r.primary-key-N</code>	Required. The last primary key column value, referenced by column name or column number.	N (N+1 if username is referenced)

## 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, use direct row handling.

This script may be ignored if none of the following scripts are defined: `upload_new_row_insert`, `upload_old_row_insert`, and `resolve_conflict`.

## SQL Example

The following SQL script is taken from the Contact sample and can be found in `%SQLANYSAMP17%\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}' )
```

## Related Information

[Direct Row Handling \[page 558\]](#)

[Data Scripts \[page 346\]](#)

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Conflict Detection \[page 134\]](#)

[resolve\\_conflict Table Event \[page 483\]](#)

[upload\\_delete Table Event \[page 500\]](#)

[upload\\_insert Table Event \[page 506\]](#)

[upload\\_update Table Event \[page 522\]](#)

[FROM Clause](#)

### 1.12.2.58 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. Use named parameters instead. 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	Description	Order (deprecated for SQL)
<code>s.remote_id</code>	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
<code>s.username</code>	VARCHAR(128). The MobiLink user name. This parameter is optional.	Optional
<code>s.script_version</code>	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.	Not applicable
<code>r.pk-column-1</code>	Required. The first primary key column value, referenced by column name or column number.	1 (2 if username is referenced)
...	...	...
<code>r.pk-column-N</code>	Required. The last primary key column value, referenced by column name or column number.	N (N+1 if username is referenced)

## 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.

### **i** Note

Conflict detection is usually performed much faster when done all at once in the `upload_update` script.

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 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, use direct row handling.

## Related Information

[Direct Row Handling \[page 558\]](#)

[Data Scripts \[page 346\]](#)

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Conflict Detection \[page 134\]](#)

[upload\\_fetch Table Event \[page 502\]](#)

[resolve\\_conflict Table Event \[page 483\]](#)

[upload\\_delete Table Event \[page 500\]](#)

[upload\\_insert Table Event \[page 506\]](#)

[upload\\_update Table Event \[page 522\]](#)

### 1.12.2.59 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. Use named parameters instead. 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	Description	Order (deprecated for SQL)
<code>s.remote_id</code>	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
<code>s.username</code>	VARCHAR(128). The MobiLink user name. This parameter is optional.	Not applicable
<code>s.script_version</code>	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.	Not applicable
<code>r.column-1</code>	Required. The first inserted column value, referenced by column name or column value.	1
...	...	...
<code>r.column-N</code>	Required. The last inserted column value, referenced by column name or column value.	N

## 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, use direct row handling.

## SQL Example

This example handles inserts that were made on the Customer table in the remote database. The script inserts the values into a table named Customer in the consolidated database. The final column of the table identifies the Customer as active. The final column does not appear in the remote database.

```
CALL ml_add_table_script(
  'ver1',
  'Customer',
  'upload_insert',
  'INSERT INTO Customer(
    cust_id,
```

```

name,
rep_id,
active )
VALUES (
{ml r.cust_id},
{ml r.name},
{ml r.rep_id},
1 )' );

```

## Related Information

[Direct Row Handling \[page 558\]](#)

[Data Scripts \[page 346\]](#)

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[upload\\_delete Table Event \[page 500\]](#)

[upload\\_update Table Event \[page 522\]](#)

[upload\\_fetch Table Event \[page 502\]](#)

### 1.12.2.60 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. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.username	VARCHAR(128). The MobiLink user name. This parameter is optional.	Optional (1 if referenced)

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
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.	Not applicable
r.column-1	Required. The first column value from the new (post-image) row, referenced by column name or column number.	1 (2 if username is referenced)
...	...	...
r.column-N	Required. The last column value from the new (post-image) row, referenced by column name or column number.	N (N+1 if username is referenced)

## Default Action

None.

## Remarks

When a MobiLink client sends an updated row to the MobiLink server, it includes not only the new values (the post-image), but also a copy of the old row values (the pre-image). When the pre-image does not match the current values in the consolidated database, a conflict is detected.

After MobiLink detects a conflict, this event allows you to save post-image values to a table. You can use this event to assist in developing conflict resolution procedures for updates. The parameters for this event hold new row values from the remote database before the update is performed on the corresponding consolidated database table. This event is also used to insert rows in forced-conflict mode (forced-conflict mode has been deprecated.).

### **i** Note

Conflict detection is usually performed much faster when done all at once in the upload\_update script.

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, use direct row handling.

## 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' )' )
```

## Related Information

[Direct Row Handling \[page 558\]](#)

[Data Scripts \[page 346\]](#)

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Conflict Handling Overview \[page 133\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[resolve\\_conflict Table Event \[page 483\]](#)

[upload\\_old\\_row\\_insert Table Event \[page 510\]](#)

[upload\\_update Table Event \[page 522\]](#)

### 1.12.2.61 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. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	N/A
s.username	VARCHAR(128). The MobiLink user name. This parameter is optional.	Optional (1 if referenced)
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.	Not applicable
<i>r.column-1</i>	Required. The first column value from the old (pre-image) row, referenced by column name or column number.	1 (2 if username is referenced)
...	...	...
<i>r.column-N</i>	Required. The last column value from the old (pre-image) row, referenced by column name or column number.	N (N+1 if username is referenced)

## Default Action

None.

## Remarks

When a MobiLink client sends an updated row to the MobiLink server, it includes not only the new values (the post-image), but also a copy of the old row values (the pre-image). When the pre-image does not match the current values in the consolidated database, a conflict is detected.

After MobiLink detects a conflict, this event allows you to save pre-image values to a table. You can use this event to assist in developing conflict resolution procedures. The parameters for this event hold old row values from the remote database before the update is performed on the corresponding consolidated database table. This event is also used to insert rows in forced-conflict mode (forced-conflict mode has been deprecated).

### **i** Note

Conflict detection is usually much faster when done all at once in the upload\_update script.

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.

## SQL Example

This example handles updates made on the product table in the remote database. The script inserts the old value of the row into a global temporary table named `product_conflict`. The final column of the table identifies the row as an old row.

```
CALL ml_add_table_script(
  'ver1',
  'table1',
  'upload_old_row_insert',
  'INSERT INTO DBA.product_conflict (
    id,
    name,
    size,
    quantity,
    unit_price,
    row_type )
VALUES (
  {ml r.id},
  {ml r.name},
  {ml r.size},
  {ml r.quantity},
  {ml r.unit_price},
  ''Old'' )' )
```

## Related Information

[Direct Row Handling \[page 558\]](#)

[Data Scripts \[page 346\]](#)

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Conflict Handling Overview \[page 133\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[resolve\\_conflict Table Event \[page 483\]](#)

[upload\\_new\\_row\\_insert Table Event \[page 508\]](#)

[upload\\_update Table Event \[page 522\]](#)



## 1.12.2.62 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.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.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.warnings	INTEGER. The number of warnings that occurred.	2
s.errors	INTEGER. The number of errors that occurred.	3
s.inserted_rows	INTEGER. The number of rows that were successfully inserted in the consolidated database.	4
s.deleted_rows	INTEGER. The number of rows that were successfully deleted from the consolidated database.	5
s.updated_rows	INTEGER. The number of rows that were successfully updated in the consolidated database.	6

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.conflicted_updates	INTEGER. The number of update rows that caused conflict. A row is included only when a resolve conflict script was successfully called for it.	9
s.ignored_inserts	INTEGER. The total number of upload insert rows that were ignored. They were ignored because 1) there is no upload_insert script in normal mode; or 2) errors occurred when the MobiLink server was invoking the corresponding script and the handle_error or handle_odbc_error event returned 1000.	10
s.ignored_deletes	INTEGER. The number of upload delete rows that caused errors while the upload_delete script was invoked, when the handle_error or handle_odbc_error are defined and returned 1000, or when there is no upload_delete script defined for the given table.	11
s.ignored_updates	INTEGER. The number of upload update rows that caused conflict but a resolve conflict script was not successfully called or no upload_update script was defined.	12
s.bytes	INTEGER. The amount of memory used within the MobiLink server to store the upload.	13
s.deadlocks	INTEGER. The number of deadlocks in the consolidated database that were detected for the synchronization.	14

## Default Action

None.

## Remarks

The upload\_statistics event allows you to gather, for any user, statistics on uploads. The upload\_statistics connection script is called just before the commit at the end of the upload transaction.

## SQL Example

The following example inserts synchronization statistics for upload operations into the table `upload_summary_audit`.

```
CALL ml_add_connection_script (
  'ver1',
  'upload_statistics',
  'INSERT INTO upload_summary_audit (
    ml_user,
    warnings,
    errors,
    inserted_rows,
    deleted_rows,
    updated_rows,
    conflicted_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`.

```
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.)

```
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 );
}}

```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)  
[MobiLink Profiler \[page 247\]](#)  
[Remote IDs and MobiLink User Names in Scripts](#)  
[SQL-Java Data Types \[page 530\]](#)  
[download\\_statistics Connection Event \[page 398\]](#)  
[download\\_statistics Table Event \[page 401\]](#)  
[upload\\_statistics Table Event \[page 517\]](#)  
[synchronization\\_statistics Connection Event \[page 487\]](#)  
[synchronization\\_statistics Table Event \[page 491\]](#)  
[time\\_statistics Connection Event \[page 494\]](#)  
[time\\_statistics Table Event \[page 497\]](#)  
[SQL-.NET Data Types \[page 546\]](#)

## 1.12.2.63 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.

In SQL scripts, you can specify event parameters by name or with a question mark. Using question marks has been deprecated. Use named parameters instead. 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	Description	Order (deprecated for SQL)
s.remote_id	VARCHAR(128). The MobiLink remote ID. You can only reference the remote ID if you are using named parameters.	Not applicable
s.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.	Not applicable
s.username	VARCHAR(128). The MobiLink user name.	1
s.table	VARCHAR(128). The table name.	2

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.warnings	INTEGER. The number of warnings issued in the upload of the table.	3
s.errors	INTEGER. The number of errors, including handled errors, that occurred in the upload of the table.	4
s.inserted_rows	INTEGER. The number of rows that were successfully inserted in the consolidated database.	5
s.deleted_rows	INTEGER. The number of rows that were successfully deleted from the consolidated database.	6
s.updated_rows	INTEGER. The number of rows that were successfully updated in the consolidated database.	7
s.conflicted_updates	INTEGER. The number of update rows that caused conflict. A row is included only when a resolve conflict script was successfully called for it.	10
s.ignored_inserts	INTEGER. The total number of upload insert rows that were ignored. They were ignored because 1) there is no upload_insert script in normal mode; or 2) errors occurred when the MobiLink server was invoking the corresponding script and the handle_error or handle_odbc_error event returned 1000.	11
s.ignored_deletes	INTEGER. The number of upload delete rows that caused errors while the upload_delete script was invoked, when the handle_error or handle_odbc_error are defined and returned 1000, or when there is no upload_delete script defined for the given table.	12
s.ignored_updates	INTEGER. The number of upload update rows that caused conflict but a resolve conflict script was not successfully called or no upload_update script was defined.	13

Parameter name for SQL scripts	Description	Order (deprecated for SQL)
s.bytes	INTEGER. The amount of memory used within the MobiLink server to store the upload.	14
s.deadlocks	INTEGER. The number of deadlocks in the consolidated database that were detected for the synchronization.	15

## Default Action

None.

## Remarks

The upload\_statistics event allows you to gather, for any user, vital statistics on synchronization happenings as they apply to any table. The upload\_statistics table script is called just before the commit at the end of the upload transaction.

### i 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.

## 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 conflictedInserts,  
            int conflictedDeletes,  
            int conflictedUpdates,  
            int ignoredInserts,  
            int ignoredDeletes,  
            int ignoredUpdates,  
            int bytes,  
            int deadlocks ) {  
            java.lang.System.out.println( "updated rows: " +  
                updatedRows );  
        }  
    }  
}
```

```
int conflictUpdates,  
int ignoredInserts,  
int ignoredDeletes,  
int ignoredUpdates,  
int bytes,  
int deadlocks ) {  
System.Console.WriteLine( "updated rows: " +  
updatedRows );  
}}}
```

## Related Information

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[MobiLink Profiler \[page 247\]](#)

[Remote IDs and MobiLink User Names in Scripts](#)

[SQL-Java Data Types \[page 530\]](#)

[download\\_statistics Connection Event \[page 398\]](#)

[upload\\_statistics Connection Event \[page 513\]](#)

[synchronization\\_statistics Connection Event \[page 487\]](#)

[synchronization\\_statistics Table Event \[page 491\]](#)

[time\\_statistics Connection Event \[page 494\]](#)

[time\\_statistics Table Event \[page 497\]](#)

[SQL-.NET Data Types \[page 546\]](#)

### 1.12.2.64 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. Use named parameters instead. 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	Order (deprecated for SQL)
<code>s.script_version</code>	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.	Not applicable
<code>r.column-1</code>	Required. The first non-primary key column value from the new (post-image) column value, referenced by column name or column number.	1
...	...	...
<code>r.column-M</code>	Required. The last non-primary key column value from the new (post-image) column value, referenced by column name or column number.	M
<code>r.pk-column-1</code>	Required. The first primary key column value from the new (post-image) column value, referenced by column name or column number.	M + 1
...	...	...
<code>r.pk-column-N</code>	Required. The last primary key column value from the new (post-image) column value, referenced by column name or column number.	M + N
<code>o.column-N</code>	Optional. The first non-primary key column value from the old (pre-image) column value, referenced by column name or column number.	M + N + 1
...	...	...
<code>o.column-M</code>	Optional. The last non-primary key column value from the old (pre-image) column value, referenced by column name or column number.	M + N + M
<code>s.script_version</code>	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.	Not applicable

## 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, use direct row handling.

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, use direct row handling.

## SQL Example

This example handles updates made to the Customer table in the remote database. The script updates the values in a table named Customer in the consolidated database.

```
CALL ml_add_table_script(
  'ver1',
  'table1',
  'upload_update',
  'UPDATE Customer
   SET name = {ml r.name}, rep_id = {ml r.rep_id}
   WHERE cust_id = {ml o.cust_id}')
```

This next example performs a similar update, but uses the old (pre-image) values to ensure the update only happens if there is no conflict. If there is a conflict, the update is ignored in this "first in wins" conflict resolution policy.

```
CALL ml_add_table_script(
  'ver1',
```

```
'table1',
'upload_update',
'UPDATE Customer
  SET name = {ml r.name}, rep_id = {ml r.rep_id}
  WHERE cust_id = {ml o.cust_id}
     AND name = {ml o.name}
     AND rep_id = {ml o.rep_id}')
```

## Related Information

[Direct Row Handling \[page 558\]](#)

[Data Scripts \[page 346\]](#)

[Script Parameters \[page 294\]](#)

[Script Additions and Deletions \[page 316\]](#)

[Conflict Detection with upload\\_update Scripts \[page 137\]](#)

[Conflict Resolution with upload\\_update Scripts \[page 139\]](#)

[upload\\_delete Table Event \[page 500\]](#)

[upload\\_fetch Table Event \[page 502\]](#)

[upload\\_insert Table Event \[page 506\]](#)

## 1.13 MobiLink Server APIs

MobiLink synchronization scripts can be written in SQL, in Java (using the MobiLink server API for Java) or in .NET (using the MobiLink server API for .NET).

### In this section:

[Synchronization Script Writing in Java \[page 526\]](#)

You control the actions of the MobiLink server by writing synchronization scripts. You can implement these scripts in SQL or Java.

[MobiLink Server Java API Reference \[page 540\]](#)

MobiLink server Java API topics explain interfaces and classes, and their associated methods and constructors. To use these classes, reference the `mlscript.jar` assembly, located in `%SQLANY17%\java\`.

[Synchronization Scripts in Microsoft .NET \[page 541\]](#)

MobiLink supports Microsoft Visual Studio programming languages for writing synchronization scripts.

[MobiLink Server .NET API Reference \[page 557\]](#)

The MobiLink .NET API topics explain interfaces and classes, and their associated methods, properties, and constructors.

[Direct Row Handling \[page 558\]](#)

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.

## 1.13.1 Synchronization Script Writing in Java

You control the actions of the MobiLink server by writing synchronization scripts. You can implement these scripts in SQL 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.

### In this section:

#### [Setting up Java Synchronization Logic \[page 526\]](#)

Use the following procedure to implement synchronization scripts in Java.

#### [Java Synchronization Logic \[page 528\]](#)

Writing Java synchronization logic requires knowledge of MobiLink events, some knowledge of Java, and knowledge of the MobiLink server API for Java.

#### [Java Synchronization Example \[page 536\]](#)

Java synchronization logic works with MobiLink and common Java classes to provide you with flexibility in deploying applications using MobiLink server.

## Related Information

[Options for Writing Server-side Synchronization Logic](#)

[Synchronization Scripts \[page 288\]](#)

[MobiLink Server Java API Reference](#)

### 1.13.1.1 Setting up Java Synchronization Logic

Use the following procedure to implement synchronization scripts in Java.

## Context

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 `%SQLANY17%\Java\mlscript.jar`.

## Procedure

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.

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.

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 17\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')
```

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 `mlsrv17 -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:

```
mlsrv17 -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.

- 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.

5. To load a specific JRE on UNIX or Linux, set the `LD_LIBRARY_PATH` (`LIBPATH` on IBM AIX (deprecated), `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.

## Related Information

[Java Methods \[page 531\]](#)

[Constructors \[page 531\]](#)

[MobiLink Server System Procedures \[page 582\]](#)

[Java Synchronization Logic \[page 528\]](#)

[Options for Writing Server-side Synchronization Logic](#)

[Synchronization Scripts \[page 288\]](#)

[ml\\_add\\_java\\_connection\\_script System Procedure \[page 593\]](#)

[ml\\_add\\_java\\_table\\_script System Procedure \[page 595\]](#)

[-sl java mlsrv17 Option \[page 84\]](#)

[Java Synchronization Example \[page 536\]](#)

[MobiLink Server Java API Reference](#)

### 1.13.1.2 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.

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 API for Java for direct access to synchronized data.

### In this section:

[Class Instances \[page 529\]](#)

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.

[Transactions \[page 530\]](#)

The normal rules regarding transactions apply to Java methods. The start and duration of database transactions is critical to the synchronization process.

[SQL-Java Data Types \[page 530\]](#)



The following table shows SQL data types and the corresponding Java data types.

#### [Constructors \[page 531\]](#)

The constructor of your class may have one of two possible signatures.

#### [Java Methods \[page 531\]](#)

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.

#### [Java Class Debugging \[page 532\]](#)

MobiLink provides various information and facilities that you may find helpful when debugging your Java code.

#### [MobiLink Server Error Handling in Java \[page 534\]](#)

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.

#### [User-Defined Start Classes in Java \[page 535\]](#)

You can define start classes that are loaded automatically when the server is started.

## Related Information

[Direct Row Handling \[page 558\]](#)

[MobiLink Server Java API Reference](#)

### 1.13.1.2.1 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.

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 might 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.

## Related Information

[Constructors \[page 547\]](#)

## 1.13.1.2.2 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.

### Related Information

[MobiLink Server Java API Reference](#)

## 1.13.1.2.3 SQL-Java Data Types

The following table shows SQL data types and the corresponding Java data types.

SQL data type	Corresponding Java data type
VARCHAR	java.lang.String
CHAR	java.lang.String
INTEGER	int or Integer
BINARY	byte[ ]
BIGINT	long
TIMESTAMP	java.sql.Timestamp
INOUT INTEGER	com.sap.ml.script.InOutInteger
INOUT VARCHAR	com.sap.ml.script.InOutString
INOUT CHAR	com.sap.ml.script.InOutString
INOUT BYTEARRAY	com.sap.ml.script.InOutByteArray

SQL data type	Corresponding Java data type
INOUT TIMESTAMP	java.sql.Timestamp

The MobiLink server automatically adds the `com.sap.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 `%SQLANY17%\java\mlscript.jar`.

## Related Information

[MobiLink Server Java API Reference](#)

### 1.13.1.2.4 Constructors

The constructor of your class may have one of two possible signatures.

```
public MyScriptClass (com.sap.ml.script.DBConnectionContext sc)
```

or

```
public MyScriptClass ()
```

The synchronization context passed to you is for the connection through which the MobiLink server is synchronizing the current user.

The `DBConnectionContext.getConnection` method returns the same database connection that MobiLink is using to synchronize the present user. You can execute statements on this connection, but you must not commit or roll back the transaction. The MobiLink server manages the transactions.

The MobiLink server prefers to use constructors with the first signature. It only uses the non-argument constructor if a constructor with the first signature is not present.

## Related Information

[MobiLink Server Java API Reference](#)

### 1.13.1.2.5 Java 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.

All methods from all events must return void.

## Related Information

[Script Additions and Deletions \[page 316\]](#)

[MobiLink Server Java API Reference](#)

### 1.13.1.2.6 Java Class Debugging

MobiLink provides various information and facilities that you may find helpful when debugging your Java code.

#### 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 17.
- 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 mlsrv17 command line option `-verbose (-v)` can be used.

## 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 mlsrv17 command line.

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.

### **i** 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.

## Related Information

[-v mlsrv17 Option \[page 92\]](#)

[-sl java mlsrv17 Option \[page 84\]](#)

[MobiLink Server Java API Reference](#)

## 1.13.1.2.7 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.

```
class TestLogListener implements LogListener {
    FileOutputStream _out_file;
    public TestLogListener(FileOutputStream out_file) {
        _out_file = out_file;
    }
    public void messageLogged(ServerContext sc, LogMessage msg) {
        String type;
        String user;
        try {
            if (msg.getType() == LogMessage.ERROR) {
                type = "ERROR";
            } else if (msg.getType() == LogMessage.WARNING) {
                type = "WARNING";
            }
            else {
                type = "UNKNOWN!!!";
            }
            user = msg.getUser();
            if (user == null) {
                user = "NULL";
            }
            _out_file.write(("Caught msg type="
                + type
                + " user=" + user
                + " text=" + msg.getText()
                + "\n").getBytes()
            );
            _out_file.flush();
        }
        catch(Exception e) {
            // Print some error output to the MobiLink log.
            e.printStackTrace();
        }
    }
}
```

The following code registers TestLogListener to receive warning messages. Call this code from anywhere that has access to the ServerContext such as a class constructor or synchronization script.

```
// ServerContext serv_context;
serv_context.addWarningListener(
    new MyLogListener(ll_out_file)
);
```

## Related Information

[MobiLink Server Java API Reference](#)

## 1.13.1.2.8 User-Defined Start Classes in Java

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 mlsrv17 -sl java option. For example, the following is part of a mlsrv17 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 com.sap.ml.script.ServerContext.

The names of loaded start classes are output to the MobiLink log with the message "Loaded JAVA start class: `classname`".

### Example

The following is a start class template. It starts a daemon thread that processes events and creates a database connection. (Not all start classes need to create a thread but if a thread is spawned it should be a daemon thread.)

```
import com.sap.ml.script.*;
import java.sql.*;
public class StartTemplate extends
    Thread implements ShutdownListener {
    ServerContext    _sc;
    Connection      _conn;
    boolean         _exit_loop;
    public StartTemplate(ServerContext sc) throws SQLException {
        // Perform setup first so that an exception
        // causes MobiLink startup to fail.
        _sc = sc;
        // Create a connection for use later.
        _conn = _sc.makeConnection();
        _exit_loop = false;
        setDaemon(true);
        start();
    }
    public void run() {
        _sc.addShutdownListener(this);
        // run() cannot throw exceptions.
        try {
            handlerLoop();
            _conn.close();
            _conn = null;
        }
        catch(Exception e) {

            // Print some error output to the MobiLink log.
            e.printStackTrace();
        }
    }
}
```

```

        // This thread shuts down and so does not
        // need to be notified of shutdown.
        _sc.removeShutdownListener(this);

        // Ask server to shutdown so that this fatal
        // error is fixed.
        _sc.shutdown();
    }
    // Shortly after return this thread no longer exists.
    return;
}

// stop our event handler loop
public void shutdownPerformed(ServerContext sc) {
    _exit_loop = true;
    try {
        // Wait max 10 seconds for thread to die.
        join(10*1000);
    }
    catch(Exception e) {
        // Print some error output to the MobiLink log.
        e.printStackTrace();
    }
}

private void handlerLoop() throws InterruptedException {
    while (!_exit_loop) {
        // Handle events in this loop. Sleep not
        // needed, block on event queue.
        sleep(1 * 1000);
    }
}
}
}
}

```

## Related Information

[-sl java mlsrv17 Option \[page 84\]](#)

[MobiLink Server Java API Reference](#)

### 1.13.1.3 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 example introduces you to this extended range of functionality using 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.
- Create a list of associated Java classes that are called during the running of your Java class.



- Store your Java classes in a location that is in the classpath for MobiLink server.

## Plan

The Java synchronization logic for this example points to the associated Java files and classes that contain functionality needed for the example to work. It shows you how to create a class `CustEmpScripts`. It shows you how to set up a synchronization context for the connection. Finally, the example provides Java methods to

- authenticate a MobiLink user.
- perform download and upload operations using cursors for each database table.

## Schema

The tables to be synchronized are `emp` and `cust`. The `emp` table has three columns called `emp_id`, `emp_name` and `manager`. The `cust` table has three columns called `cust_id`, `cust_name` and `emp_id`. All columns in each table are synchronized. The mapping from consolidated to remote database is such that the table names and column names are identical in both databases. One additional table, an audit table, is added to the consolidated database.

## Java Class Files

The files used in the example are included in the `Samples\MobiLink\JavaAuthentication` directory.

## Setup

The following code sets up the Java synchronization logic. The import statements tell the Java VM the location of needed files. The public class statement declares the class.

```
// Use a package when you create your own script.
import com.sap.ml.script.InOutInteger;
import com.sap.ml.script.DBConnectionContext;
import com.sap.ml.script.ServerContext;
import java.sql.*;
public class CustEmpScripts {
    // Context for this synchronization connection.
    DBConnectionContext _conn_context;
    // Same connection MobiLink uses for sync.
    // Do not commit or close this.
    Connection _sync_connection;
    Connection _audit_connection;
    //Get a user id given the user name. On audit connection.
    PreparedStatement _get_user_id_pstmt;
    // Add record of user logins added. On audit connection.
    PreparedStatement _insert_login_pstmt;
    // Prepared statement to add a record to the audit table.
```

```

// On audit connection.
PreparedStatement _insert_audit_pstmt;

// ...
}

```

The CustEmpScripts constructor sets up all the prepared statements for the authenticateUser method. It sets up member data.

```

public CustEmpScripts(DBConnectionContext cc) throws SQLException {
    try {
        _conn_context = cc;
        _sync_connection = _conn_context.getConnection();
        ServerContext serv_context =
            _conn_context.getServerContext();
        _audit_connection = serv_context.makeConnection();
        // Get the prepared statements ready.
        _get_user_id_pstmt =
            _audit_connection.prepareStatement(
                "select user_id from ml_user where name = ?"
            );
        _insert_login_pstmt =
            _audit_connection.prepareStatement(
                "INSERT INTO login_added(ml_user, add_time) "
                + "VALUES (?, { fn CONVERT({ fn NOW() }, SQL_VARCHAR) })"
            );
        _insert_audit_pstmt =
            _audit_connection.prepareStatement(
                "INSERT INTO login_audit(ml_user_id, audit_time, audit_action) "
                + "VALUES (?, { fn CONVERT({ fn NOW() }, SQL_VARCHAR) }, ?)"
            );
    }
    catch(SQLException e) {
        freeJDBCResources();
        throw e;
    }
    catch(Error e) {
        freeJDBCResources();
        throw e;
    }
}

```

The finalize method cleans up JDBC resources if end\_connection is not called. It calls the freeJDBCResources method, which frees allocated memory and closes the audit connection.

```

protected void finalize() throws SQLException, Throwable {
    super.finalize();
    freeJDBCResources();
}
private void freeJDBCResources() throws SQLException {
    if (_get_user_id_pstmt != null) {
        _get_user_id_pstmt.close();
    }
    if (_insert_login_pstmt != null) {
        _insert_login_pstmt.close();
    }
    if (_insert_audit_pstmt != null) {
        _insert_audit_pstmt.close();
    }
    if (_audit_connection != null) {
        _audit_connection.close();
    }
    _conn_context = null;
    _sync_connection = null;
    _audit_connection = null;
    _get_user_id_pstmt = null;
}

```

```

        _insert_login_pstmt = null;
        _insert_audit_pstmt = null;
    }

```

The endConnection method cleans up resources once the resources are not needed.

```

public void endConnection() throws SQLException {
    freeJDBCResources();
}

```

The authenticateUser method below approves all user logins and logs user information to database tables. If the user is not in the ml\_user table they are logged to login\_added. If the user id is found in ml\_user then they are logged to login\_audit. In a real system you would not ignore the user\_password, but this sample approves all users for simplicity. The endConnection method throws SQLException if any of the database operations fail with an exception.

```

public void authenticateUser(
    InOutInteger authentication_status,
    String user_name) throws SQLException
{
    boolean new_user;
    int user_id;
    // Get ml_user id.
    _get_user_id_pstmt.setString(1, user_name);
    ResultSet user_id_rs =
    _get_user_id_pstmt.executeQuery();
    new_user = !user_id_rs.next();
    if (!new_user) {
        user_id = user_id_rs.getInt(1);
    }
    else {
        user_id = 0;
    }

    user_id_rs.close();
    user_id_rs = null;
    // In this tutorial always allow the login.
    authentication_status.setValue(1000);

    if (new_user) {
        _insert_login_pstmt.setString(1, user_name);
        _insert_login_pstmt.executeUpdate();
        java.lang.System.out.println("user: " + user_name + " added. ");
    }
    else {
        _insert_audit_pstmt.setInt(1, user_id);
        _insert_audit_pstmt.setString(2, "LOGIN ALLOWED");
        _insert_audit_pstmt.executeUpdate();
    }
    _audit_connection.commit();
    return;
}

```

The following methods use SQL statements to act as cursors on the database tables. Since these are cursor scripts, they must return a SQL string.

```

public static String empUploadInsertStmt() {
    return("INSERT INTO emp(emp_id, emp_name) VALUES(?, ?)");
}
public static String empUploadDeleteStmt() {
    return("DELETE FROM emp WHERE emp_id = ?");
}
public static String empUploadUpdateStmt() {
    return("UPDATE emp SET emp_name = ? WHERE emp_id = ?");
}

```

```

}
public static String empDownloadCursor() {
    return("SELECT emp_id, emp_name FROM emp");
}
public static String custUploadInsertStmt() {
    return("INSERT INTO cust(cust_id, emp_id, cust_name) VALUES (?, ?, ?)");
}
public static String custUploadDeleteStmt() {
    return("DELETE FROM cust WHERE cust_id = ?");
}
public static String custUploadUpdateStmt() {
    return("UPDATE cust SET emp_id = ?, cust_name = ? WHERE cust_id = ?");
}
public static String custDownloadCursor() {
    return("SELECT cust_id, emp_id, cust_name FROM cust");
}
}

```

Use the following command to compile the code:

```
javac -cp %sqlany17%\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:

```
mlsrv17 ... -sl java (-cp <class_location>)
```

## Related Information

[Setting up Java Synchronization Logic \[page 526\]](#)

[MobiLink Server Java API Reference](#)

## 1.13.2 MobiLink Server Java API Reference

MobiLink server Java API topics explain interfaces and classes, and their associated methods and constructors. To use these classes, reference the `mlscript.jar` assembly, located in `%SQLANY17%\java\`.

### Package

```
com.sap.ml.script
```

The MobiLink server Java API reference is available in the *MobiLink - Java API Reference* at <https://help.sap.com/viewer/c2c7ac13c75942d183ab7e8f6b459eda/LATEST/en-US>.

## Related Information

[MobiLink Server Java API Reference](#)

### 1.13.3 Synchronization Scripts in Microsoft .NET

MobiLink supports Microsoft Visual Studio programming languages for writing synchronization scripts.

To write MobiLink scripts targeting the Microsoft .NET Framework, you can use any language that lets you create valid Microsoft .NET assemblies. In particular, the following languages are tested and documented:

- Microsoft Visual C#
- Microsoft Visual Basic
- Microsoft Visual C++

Synchronization logic written in a Microsoft .NET language can operate in the same manner as logic written in SQL: on the occurrence of an event, the MobiLink server can make a call to a user-defined method, instead of running a SQL script. The method can return a SQL string to MobiLink.

#### In this section:

##### [Implementing Synchronization Scripts in .NET \[page 542\]](#)

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.

##### [.NET Synchronization Logic \[page 544\]](#)

To write .NET synchronization logic, you require knowledge of MobiLink events, some knowledge of .NET, and familiarity with the MobiLink server API for .NET.

##### [.NET Synchronization Techniques \[page 553\]](#)

There are techniques you can use to tackle common .NET synchronization tasks.

##### [.NET Assembly Loading \[page 553\]](#)

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`.

##### [.NET Synchronization Example \[page 556\]](#)

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.

## Related Information

[Options for Writing Server-side Synchronization Logic](#)

[Synchronization Scripts \[page 288\]](#)

[MobiLink Server .NET API Reference \[page 557\]](#)

## 1.13.3.1 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.

### Procedure

1. Create your own class or classes. Write a method for each required synchronization event. These methods must be public.

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.

2. Create one or more assemblies. While compiling, reference `Sap.MobiLink.Script.dll`, which contains a repository of MobiLink server API classes to use in your own .NET methods.

`Sap.MobiLink.Script.dll` is located in `%SQLANY17%\Assembly\V3.5`.

You can compile your class on the command line, or using Microsoft Visual Studio or another .NET development environment.

3. Compile your project.

For example, compile from Microsoft Visual Studio as follows:

- a. On the *VS.NET Project* menu, click *Add Existing Item*.
- b. Locate `Sap.MobiLink.Script.dll`.

In the *Open* list, click *Link File*.

#### Note

For Microsoft Visual Studio, always use the Link File method. Do not use the Add Reference option to reference `Sap.MobiLink.Script.dll`. The Add Reference option duplicates `Sap.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 `Sap.MobiLink.Script.dll`. for example, in C#:

```
csc /out:dll-path\out.dll /target:library /reference:dll-path\Sap.MobiLink.Script.dll sync_v1.cs
```

4. In the MobiLink system tables in your consolidated database, specify the name of the package, class, and method to call for each synchronization script. No more than one class is permitted per script version.

For example, you can add this information to the MobiLink system tables using the `ml_add_dnet_connection_script` stored procedure or the `ml_add_dnet_table_script` stored procedure. The following SQL statement, when run in a SQL Anywhere database, specifies that `myNamespace.myClass.myMethod` should be run whenever the `authenticate_user` connection-level event occurs.

```
CALL ml_add_dnet_connection_script(
```

```
'version1',  
'authenticate_user',  
'myNamespace.myClass.myMethod'  
)
```

### Note

The fully qualified method name is case sensitive.

As a result of this procedure call, the `script_language` column of the `ml_script` system table contains the word `dnet`. The `script` column contains the qualified name of a public .NET method.

You can also add this information using SQL Central.

5. Instruct the MobiLink server to load assemblies and start the CLR. You tell MobiLink where to locate these assemblies using options in the `mlsrv17` 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:

```
mlsrv17 -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.

#### 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.

### Note

You can use the `-MLAutoLoadPath` option or the `-MLDomConfigFile` option, but not both.

## Results

The .NET synchronization logic is set up.

## Related Information

[.NET Methods \[page 547\]](#)

[Constructors \[page 547\]](#)

[Script Additions and Deletions \[page 316\]](#)

[.NET Assembly Loading \[page 553\]](#)

[MobiLink Server .NET API Reference \[page 557\]](#)

[ml\\_add\\_dnet\\_connection\\_script System Procedure \[page 591\]](#)

[ml\\_add\\_dnet\\_table\\_script System Procedure \[page 592\]](#)

[-sl dnet mlsrv17 Option \[page 82\]](#)

### 1.13.3.2 .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.

.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.

#### In this section:

##### [Class Instances \[page 545\]](#)

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.

##### [Transactions \[page 546\]](#)

The normal rules regarding transactions apply to .NET methods.

##### [SQL-.NET Data Types \[page 546\]](#)

The following table shows SQL data types and the corresponding .NET data types for MobiLink script parameters.

##### [Constructors \[page 547\]](#)

The constructor of your class takes no parameters or takes one `Sap.MobiLink.Script.DBConnectionContext` parameter.

##### [.NET Methods \[page 547\]](#)

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.

##### [User-Defined Start Classes in .NET \[page 548\]](#)

You can define start classes that are loaded automatically when the server is started.

##### [How to Print Information From .NET \[page 550\]](#)

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.

##### [MobiLink Server Error Handling With .NET \[page 550\]](#)



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.

#### [Setting Break Points to Debug .NET Synchronization Logic \[page 551\]](#)

The following procedure can be used to set break points so you can debug your .NET scripts using Microsoft Visual Studio.

#### [Debugging .NET Synchronization Logic \[page 552\]](#)

The following procedure can be used to debug your .NET scripts using Microsoft Visual Studio.

## Related Information

[Direct Row Handling \[page 558\]](#)

[MobiLink Server .NET API Reference \[page 557\]](#)

### 1.13.3.2.1 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.

#### **i** 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 might 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.

## Related Information

[Constructors \[page 547\]](#)

[MobiLink Server .NET API Reference \[page 557\]](#)

## 1.13.3.2 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.

### Related Information

[MobiLink Server .NET API Reference \[page 557\]](#)

## 1.13.3.2.3 SQL-.NET Data Types

The following table shows SQL data types and the corresponding .NET data types for MobiLink script parameters.

SQL data type	Corresponding .NET data type
VARCHAR	string
CHAR	string
INTEGER	int
BIGINT	long
BINARY	byte [ ]
TIMESTAMP	DateTime
INOUT INTEGER	ref int
INOUT VARCHAR	ref string
INOUT CHAR	ref string
INOUT BYTEARRAY	ref byte [ ]
INOUT TIMESTAMP	ref DateTime

## Related Information

[MobiLink Server .NET API Reference \[page 557\]](#)

### 1.13.3.2.4 Constructors

The constructor of your class takes no parameters or takes one `Sap.MobiLink.Script.DBConnectionContext` parameter.

For example:

```
public ExampleClass(Sap.MobiLink.Script.DBConnectionContext cc)
```

or

```
public ExampleClass()
```

The synchronization context passed to you is for the connection through which the MobiLink server is synchronizing the current user.

The `DBConnectionContext.GetConnection` method returns the same database connection that MobiLink is using to synchronize the present user. You can execute statements on this connection, but you must not commit or roll back the transaction. The MobiLink server manages the transactions.

The MobiLink server uses the constructor that takes an `Sap.MobiLink.Script.DBConnectionContext` parameter if it exists. If it does not, it uses the void constructor.

## Related Information

[MobiLink Server .NET API Reference \[page 557\]](#)

### 1.13.3.2.5 .NET 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.

All methods from all events must return void.

## Related Information

[MobiLink Server .NET API Reference \[page 557\]](#)

### 1.13.3.2.6 User-Defined Start Classes in .NET

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 mlsrv17 -sl dnet option. For example, the following is part of an mlsrv17 command line. It causes mycl1 and mycl2 to be loaded as start classes.

```
-sl dnet (-MLStartClasses=MyNameSpace.MyClass.mycl1,MyNameSpace.MyClass.mycl2)
```

Classes are loaded in the order in which they are listed. If the same class is listed more than once, more than one instance is created.

All start classes must be public and must have a public constructor that either accepts no arguments or accepts one argument of type MobiLink.Script.ServerContext.

The names of loaded start classes are output to the MobiLink log with the message "Loaded .NET start class: `classname`".

Use the GetStartClassInstances method to see the start classes that are constructed at server start time.

## 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 Sap.MobiLink.Script;
namespace TestScripts {
    public class MyStartClass {
        ServerContext    _sc;
        bool              _exit_loop;
        Thread            _thread;
        OdbcConnection   _conn;
    }
}
```

```

public MyStartClass(ServerContext sc) {
    // Perform setup first so that an exception
    // causes MobiLink startup to fail.
    _sc = sc;
    // Create connection for use later.
    _conn = _sc.makeConnection();
    _exit_loop = false;
    _thread = new Thread(new ThreadStart(run)) ;
    _thread.IsBackground = true;
    _thread.Start();
}
public void run() {
    ShutdownCallback callback = new ShutdownCallback(shutdownPerformed);
    _sc.ShutdownListener += callback;
    // run() can't throw exceptions.
    try {
        handlerLoop();
        _conn.close();
        _conn = null;
    }
    catch(Exception e) {
        // Print some error output to the MobiLink log.
        Console.Error.Write(e.ToString());

        // There is no need to be notified of shutdown.
        _sc.ShutdownListener -= callback;
        // Ask server to shut down so this fatal error can be fixed.
        _sc.Shutdown();
    }
    // Shortly after return, this thread no longer exists.
    return;
}
public void shutdownPerformed(ServerContext sc) {
    // Stop the event handler loop.
    try {
        _exit_loop = true;

        // Wait a maximum of 10 seconds for thread to die.
        _thread.Join(10*1000);
    }
    catch(Exception e) {
        // Print some error output to the MobiLink log.
        Console.Error.Write(e.ToString());
    }
}
private void handlerLoop() {
    while (!_exit_loop) {
        // Handle events in this loop.
        Thread.Sleep(1*1000);
    }
}
}
}
}

```

## Related Information

[-sl dnet mlsrv17 Option \[page 82\]](#)

[MobiLink Server .NET API Reference \[page 557\]](#)

## 1.13.3.2.7 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.

### i 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 about how your scripts are used.

## Related Information

[MobiLink Server .NET API Reference \[page 557\]](#)

## 1.13.3.2.8 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;
        }
        _output_file.WriteLine("Caught msg type=" + type
            + " user=" + user
            + " text=" + lm.Text);
        _output_file.Flush();
    }
}
```

```
}  
    StreamWriter _output_file;  
}
```

The following code registers the TestLogListener. Call this code from somewhere that has access to the ServerContext such as a class constructor or synchronization script.

```
// ServerContext serv_context;  
TestLogListener errttl = new TestLogListener(log_listener_file);  
serv_context.ErrorListener += new LogCallback(errttl.errCallback);
```

## Related Information

[MobiLink Server .NET API Reference \[page 557\]](#)

### 1.13.3.2.9 Setting Break Points to Debug .NET Synchronization Logic

The following procedure can be used to set break points so you can debug your .NET scripts using Microsoft Visual Studio.

#### Procedure

1. Start Microsoft Visual Studio.
2. Click **Tools** > **Attach to Process**.
3. In the *Available Processes* control, select *mlsrv17.exe* and then press *Attach*.
4. Set your break points.
5. Start a synchronization.

#### Results

The script can be debugged.

## Related Information

[-sl dnet mlsrv17 Option \[page 82\]](#)

[MobiLink Server .NET API Reference \[page 557\]](#)

## 1.13.3.2.10 Debugging .NET Synchronization Logic

The following procedure can be used to debug your .NET scripts using Microsoft Visual Studio.

### Procedure

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 Microsoft Visual Studio that contain your source files.
3. In this step, you start a new Microsoft Visual Studio instance to debug the MobiLink server and your .NET synchronization scripts. Start Microsoft Visual Studio using a command line option to debug the MobiLink server.
  - At a command prompt, navigate to the `Common7\IDE` subdirectory of your Microsoft Visual Studio installation.
  - Start devenv (the Microsoft Visual Studio IDE) using the /debugexe option. For example, run the following command to debug the MobiLink server. Remember to specify mlsrv17 options, including the connection string and the option to load .NET assemblies. For 32-bit Windows environments:

```
devenv /debugexe %sqlany17%\bin32\mlsrv17.exe -c ...
```

For 64-bit Windows environments:

```
devenv /debugexe %sqlany17%\bin64\mlsrv17.exe -c ...
```

Microsoft Visual Studio starts and *mlsrv17.exe* appears in the Solution Explorer window.

4. Set up Microsoft Visual Studio for debugging .NET code:
  - In the Microsoft Visual Studio Solution Explorer window, right-click *mlsrv17.exe* and choose Properties.
  - Change Debugger Type from Auto to Mixed or Managed Only to ensure that Microsoft Visual Studio only debugs your .NET synchronization scripts. In Microsoft 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.

5. Open the associated .NET source files and set break points.

Open the source files individually in the mlsrv17 solution. Do not open the original solution or project file.



6. Start MobiLink from the Debug menu or by pressing F5.

If prompted, save `m/srv17.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.

## Related Information

[-sl dnet m/srv17 Option \[page 82\]](#)

[MobiLink Server .NET API Reference \[page 557\]](#)

### 1.13.3.3 .NET Synchronization Techniques

There are techniques you can use to tackle common .NET synchronization tasks.

To upload or download rows via .NET, use direct row handling.

## Related Information

[Direct Row Handling \[page 558\]](#)

[MobiLink Server .NET API Reference \[page 557\]](#)

### 1.13.3.4 .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 `m1DomConfig.xml`:

```
<?xml version="1.0" encoding="utf-8"?>
<config xmlns="Sap.MobiLink.m1DomConfig"
xsi:schemaLocation='Sap.MobiLink.m1DomConfig m1DomConfig.xsd' xmlns:xsi='http://
www.w3.org/2001/XMLSchema-instance' >
  <domain>
    <name>SampleDomain1</name>
    <appBase>C:\scriptsDir</appBase>
    <configFile></configFile>
    <assembly name="Assembly1" />
    <assembly name="Assembly2" />
  </domain>
  <domain>
    <name>SampleDomain2</name>
    <appBase>\Dom2assembly</appBase>
    <configFile>\Dom2assembly\AssemblyRedirects.config</configFile>
    <assembly name="Assembly3" />
    <assembly name="Assembly4" />
  </domain>
</config>
```

The following is an explanation of the contents of `m1DomConfig.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.

## Related Information

[Recommended ODBC Drivers For MobiLink](#)

[-sl dnet mlsrv17 Option \[page 82\]](#)

[MobiLink Server .NET API Reference \[page 557\]](#)

## 1.13.3.5 .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:

```
CREATE TABLE user_pwd_table (  
    user_name  varchar(128) PRIMARY KEY NOT NULL,  
    pwd        varchar(128)  
)
```

2. Add a user and password to the table:

```
INSERT INTO user_pwd_table VALUES('user1', 'myPwd')
```

3. Create a directory for your .NET assembly. For example, `c:\mlexample`.
4. Create a file called `AuthUser.cs` with the following contents:

```
using System;  
using Sap.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;  
            // Set the size for SQL_VARCHAR.  
            user_param.Size = (uint) user.Length;  
            user_param.Value = user;  
            pwd_command.Parameters.Add(user_param);  
            // Fetch the password for this user.  
            DBRowReader rr = pwd_command.ExecuteReader();  
            object[] pwd_row = rr.NextRow();  
            if (pwd_row == null) {  
                // User is unknown.  
                authStatus = 4000;  
            }  
            else {  
                if (((string) pwd_row[0]) == pwd) {  
                    // Password matched.  
                    authStatus = 1000;  
                }  
                else {
```

```

        // Password did not match.
        authStatus = 4000;
    }
}
pwd_command.Close();
rr.Close();
return;
}
}

```

The `MLEExample.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 Microsoft 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:"%SQLANY17%
\Assembly\V3.5\Sap.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 `MLEExample` and class `AuthClass`. Execute the following SQL:

```

CALL ml_add_dnet_connection_script('ex_version', 'authenticate_user',
'MLEExample.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`.

## Related Information

[authenticate\\_user Connection Event \[page 354\]](#)

[MobiLink Server .NET API Reference \[page 557\]](#)

## 1.13.4 MobiLink Server .NET API Reference

The MobiLink .NET API topics explain interfaces and classes, and their associated methods, properties, and constructors.

### Namespace

```

Sap.MobiLink.Script

```

## Remarks

To use these classes, reference the `Sap.MobiLink.Script.dll` assembly, located in `%SQLANY17%\Assembly\V3.5.`

The MobiLink server .NET API reference is available in the *MobiLink - .NET API Reference* at <https://help.sap.com/viewer/935ac449a8764285843c9cb0012d5f05/LATEST/en-US>.

## Related Information

[MobiLink Server Java API Reference](#)

### 1.13.5 Direct Row Handling

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.

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.

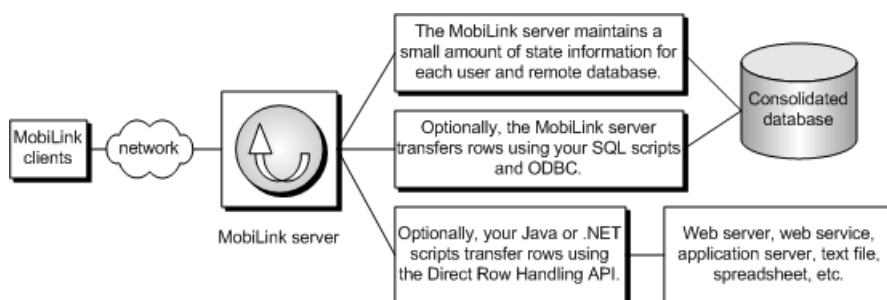
#### Direct row handling

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:



#### In this section:

##### [The Components of Direct Row Handling \[page 559\]](#)

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 Row Handling Setup \[page 560\]](#)

To use direct row handling, you need familiarity with how to create a MobiLink consolidated database, add synchronization scripts, and create MobiLink remote users.

##### [Development Tips for Direct Row Handling \[page 561\]](#)

Keep the following tips in mind when working with direct row handling.

##### [Direct Uploads \[page 563\]](#)

Following is an overview of how to handle direct uploads.

##### [Direct Upload Conflicts \[page 564\]](#)

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.

##### [Direct Downloads \[page 568\]](#)

Following is an overview of how to handle direct downloads.

## 1.13.5.1 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.

#### **handle\_DownloadData**

allows you to set download operations using the `DownloadData` interface.

## Components of the MobiLink Server API for Direct Row Handling

For the Java API:

- `DBConnectionContext.getDownloadData`
- `DownloadData` interface
- `DownloadTableData` interface
- `UpdateResultSet` interface
- `UploadData` interface
- `UploadedTableData` interface

For the .NET API:

- `DBConnectionContext.GetDownloadData` method
- `DownloadData` interface
- `DownloadTableData` interface
- `UpdateDataReader` interface
- `UploadedTableData` interface
- `UploadData` interface

## Related Information

[Direct Uploads \[page 563\]](#)

[Direct Downloads \[page 568\]](#)

[handle\\_UploadData Connection Event \[page 448\]](#)

[handle\\_DownloadData Connection Event \[page 437\]](#)

### 1.13.5.2 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.
2. To handle uploads, write a public method using the `UploadData` interface and register it for the `handle_UploadData` connection event.
3. To handle downloads, write a public method using the `DownloadData` interface and register it for the `handle_DownloadData` connection event (or another event).



## Related Information

[MobiLink Consolidated Databases \[page 152\]](#)

[Direct Uploads \[page 563\]](#)

[Direct Downloads \[page 568\]](#)

[Tutorial: Using Direct Row Handling](#)

[Setting up Java Synchronization Logic \[page 526\]](#)

[Implementing Synchronization Scripts in .NET \[page 542\]](#)

[SAP SQL Anywhere !\[\]\(003082e50e3009141f59bd5df831749f\_img.jpg\)](#)

### 1.13.5.3 Development Tips for Direct Row Handling

Keep the following tips in mind when working with 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.

#### 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.

## 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 rolls back 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.

## Related Information

[Unique Primary Keys \[page 126\]](#)

[Implementing Timestamp-based Downloads \[page 115\]](#)

## 1.13.5.4 Direct Uploads

Following is an overview of how to handle direct uploads.

1. Register a Java or .NET method for the handle\_UploadData connection event.
2. Write a method for the handle\_UploadData synchronization event. This event accepts one UploadData parameter.

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.

To create a transactional upload, use the -tu dbmlsync option.

To register connection-level events, use:

- ml\_add\_java\_connection\_script system procedure
- ml\_add\_dnet\_connection\_script system procedure

## 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.

### 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.

### 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.

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.

## Example

For an example of how to handle direct uploads, see the handle\_UploadData connection event.

## Related Information

[Synchronization Script Writing in Java \[page 526\]](#)

[Synchronization Scripts in Microsoft .NET \[page 541\]](#)

[handle\\_UploadData Connection Event \[page 448\]](#)

[-tu dbmlsync Option](#)

[ml\\_add\\_java\\_connection\\_script System Procedure \[page 593\]](#)

[ml\\_add\\_dnet\\_connection\\_script System Procedure \[page 591\]](#)

### 1.13.5.5 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.

## 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:

```
current data source - (old remote - new remote)
-> 7 - (10-6) = 3
```

The following procedures for Java and .NET demonstrate how you can resolve this conflict for direct uploads, using the following table as an example:

```
CREATE TABLE remoteOrders
(
    pk integer primary key not null,
    inventory integer not null
);
```

#### Java

1. Register a Java method for the handle\_UploadData connection event.  
For example, the following stored procedure call registers a Java method called HandleUpload for the handle\_UploadData connection event when synchronizing the script version ver1. You run this stored procedure against your MobiLink consolidated database.

```
call ml_add_java_connection_script( 'ver1',
    'handle_UploadData',
    'OrderProcessor.HandleUpload' )
```

2. Obtain an UpdateResultSet for a table in the upload.  
The OrderProcessor.HandleUpload method obtains an UpdateResultSet for the remoteOrders table:

```
// method for handle_UploadData event
public void HandleUpload( UploadData u_data )
{
    // Get UploadedTableData for the remoteOrders table.
```

```

        UploadedTableData u_table =
u_data.getUploadedTableByName("remoteOrders");

        // Get an UpdateResultSet for the remoteOrders table.
        UpdateResultSet update_rs = u_table.getUpdates();

        // (Continued...)

```

- For each update, get the current values in your central data source. In this example, the UpdateResultSet getInt method returns an integer value for the primary key column (the first column). You can implement and then use the getMyCentralData method to get data from your central data source.

```

while( update_rs.next() )
{
    // Get central data source values.
    // Get the primary key value.
    int pk_value = update_rs.getInt(1);
    // Get central data source values.
    int central_value = getMyCentralData(pk_value);
    // (Continued...)
}

```

- For each update, get the old and new values uploaded by the MobiLink client. The example uses the UpdateResultSet setOldRowValues and UpdateResultSet setNewRowValues for old and new values, respectively.

```

// Set mode for old row values.
update_rs.setOldRowValues();
// Get the _old_ stored value on the remote.
int old_value = update_rs.getInt(2);
// Set mode for new row values.
update_rs.setNewRowValues();
// Get the _new_ updated value on the remote.
int new_value = update_rs.getInt(2);
// (Continued...)

```

- For each update, check for conflicts. A conflict occurs when the old row value does not match the current value in the central data source. To resolve the conflict, a resolved value is calculated using business logic. If no conflict occurs, the central data source is updated with the new remote value. You can implement and then use the setMyCentralData method to perform the update.

```

// Check if there is a conflict.
if(old_value == central_value)
{
    // No conflict.
    setMyCentralData(pk_value, new_value);
}
else
{
    // Handle the conflict.
    int inventory = old_value - new_value;
    int resolved_value = central_value - inventory;

    setMyCentralData(pk_value, resolved_value);
}
}

```

## .NET

- Register a method for the handle\_UploadData connection event.

For example, the following stored procedure call registers a .NET method called HandleUpload for the handle\_UploadData connection event when synchronizing the script version ver1. You run this stored procedure against your MobiLink consolidated database.

```
call ml_add_dnet_connection_script( 'ver1',
    'handle_UploadData',
    'MyScripts.OrderProcessor.HandleUpload' )
```

2. Obtain an UpdateDataReader for a table in the upload.

The MyScripts.OrderProcessor.HandleUpload method obtains an UpdateResultSet for the remoteOrders table:

```
// method for handle_UploadData event
public void HandleUpload( UploadData u_data )
{
    // Get UploadedTableData for the remoteOrders table.
    UploadedTableData u_table =
u_data.GetUploadedTableByName("remoteOrders");

    // Get an UpdateDataReader for the remoteOrders table.
    UpdateDataReader update_dr = u_table.GetUpdates();

    // (Continued...)
```

3. For each update, get the current values in your central data source.

In this example, the UpdateDataReader GetInt32 method returns an integer value for the primary key column (the first column). You can implement and then use the getMyCentralData method to get data from your central data source.

```
while( update_dr.Read() )
{
    // Get central data source values.
    // Get the primary key value.
    int pk_value = update_dr.GetInt32(0);
    // Get central data source values.
    int central_value = getMyCentralData(pk_value);
    // (Continued...)
```

4. For each update, get the old and new values uploaded by the MobiLink client.

The example uses the UpdateResultSet setOldRowValues and UpdateResultSet setNewRowValues for old and new values, respectively.

```
// Set mode for old row values.
update_dr.SetOldRowValues();
// Get an _old_ value.
int old_value = update_dr.GetInt32(1);
// Set mode for new row values.
update_dr.SetNewRowValues();
// Get the _new_ updated value.
int new_value = update_dr.GetInt32(1);
// (Continued...)
```

5. For each update, check for conflicts.

A conflict occurs when the old row value does not match the current value in the central data source. To resolve the conflict, a resolved value is calculated using business logic. If no conflict occurs, the central data source is updated with the new remote value. You can implement and then use the setMyCentralData method to perform the update.

```
// Check if there is a conflict.
if(old_value == central_value)
```

```

{
    // No conflict.
    setMyCentralData(pk_value, new_value);
}
else
{
    // Handle the conflict.
    int inventory = old_value - new_value;
    int resolved_value = central_value - inventory;

    setMyCentralData(pk_value, resolved_value);
}
}

```

## Related Information

[Conflict Handling Overview \[page 133\]](#)

[Script Additions and Deletions \[page 316\]](#)

[handle\\_UploadData Connection Event \[page 448\]](#)

[ml\\_add\\_java\\_connection\\_script System Procedure \[page 593\]](#)

[ml\\_add\\_dnet\\_connection\\_script System Procedure \[page 591\]](#)

### 1.13.5.6 Direct Downloads

Following is an overview of how to handle direct downloads.

1. Register a Java or .NET method for the handle\_DownloadData connection event.
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.

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.

## 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.

### **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.

## Example

For an example of how to handle direct downloads, see the handle\_DownloadData connection event.

## Related Information

[handle\\_DownloadData Connection Event \[page 437\]](#)

[MobiLink Complete Event Model \[page 340\]](#)

## 1.14 MobiLink Reference

Many useful tools and resources are available to help you use MobiLink.

### In this section:

[MobiLink Replay C++ Callbacks \[page 570\]](#)

A complete list of callbacks that can be generated by the mlgenreplayapi utility is included in the mlreplaycallbacks.cpp file. These callbacks can be developed to customize the data uploaded to the MobiLink server during a replay session using the mlreplay utility.

[MobiLink Server System Procedures \[page 582\]](#)

MobiLink provides the following stored procedures to help you create your applications.

[MobiLink Utilities \[page 647\]](#)

A set of utility programs are included with MobiLink server. Each of the utilities can be accessed from one or more of SQL Central, Interactive SQL, or at a command prompt.

[MobiLink Data Mappings Between Remote and Consolidated Databases \[page 662\]](#)

Depending on the consolidated database you are using, the MobiLink server may map a specified data type to a different data type.

[Character Set Considerations \[page 708\]](#)

Each character of text is represented in one or more bytes. The mapping from characters to binary codes is called the **character set encoding**.

[ODBC Drivers for MobiLink \[page 710\]](#)

The MobiLink server can work with a variety of consolidated databases and ODBC drivers. Some drivers, though compatible for use with MobiLink, may have functional restrictions associated with their use.

## 1.14.1 MobiLink Replay C++ Callbacks

A complete list of callbacks that can be generated by the `mlgenreplayapi` utility is included in the `mlreplaycallbacks.cpp` file. These callbacks can be developed to customize the data uploaded to the MobiLink server during a replay session using the `mlreplay` utility.

### **i** 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.

### In this section:

#### [CreateAndInitMLReplayUploadTransaction Callback \[page 571\]](#)

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.

#### [DelayCreationOfSimulatedClient Callback \[page 572\]](#)

Can be used to coordinate when each simulated client is created based on the given simulated client number and the number of simulated clients.

#### [DelayDestructionOfSimulatedClient Callback \[page 573\]](#)

Can be used to coordinate when each simulated client is destroyed; called once per simulated client, per `mlreplay` instance.

#### [DelayStartOfReplay Callback \[page 573\]](#)

Can be used to coordinate when replaying begins; called once per repetition, per simulated client per `mlreplay` instance.

#### [DestroyMLReplayUploadTransaction Callback \[page 574\]](#)

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.

#### [FiniIdentifySimulatedClient Callback \[page 575\]](#)

Used to clean up memory used by the call to `IdentifySimulatedClient` for the given simulated client; called once per simulated client.

#### [GetDownloadApplyTime Callback \[page 576\]](#)

Used to simulate slow devices; called once per download, per simulated client, and per repetition.

#### [GetMLReplayAPIVersion Callback \[page 577\]](#)

Used to return the replay API version.

#### [GetUploadTransaction Callback \[page 577\]](#)

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.

#### [GlobalFini Callback \[page 579\]](#)

Used to clean up any global variables used by the other callbacks; called once per `mlreplay` instance.

#### [GlobalInit Callback \[page 579\]](#)

Used to initialize any global variables used by the other callbacks; called once per `mlreplay` instance.

#### [IdentifySimulatedClient Callback \[page 580\]](#)

Used to specify the simulated client information; called once per simulated client.

#### [ReportEndOfReplay Callback \[page 581\]](#)

Used to perform any actions required when a simulated client is finished replaying; called once per simulated client, and per repetition.

## Related Information

[MobiLink Replay Utility \(mlreplay\) \[page 651\]](#)

### 1.14.1.1 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

```
_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

Do not modify this callback.

## 1.14.1.2 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

```
_MLREPLAY_EXPORT bool _MLREPLAY_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.

### 1.14.1.3 DelayDestructionOfSimulatedClient Callback

Can be used to coordinate when each simulated client is destroyed; called once per simulated client, per mlreplay instance.

#### ≡ Syntax

```
_MLREPLAY_EXPORT bool _MLREPLAY_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.

### 1.14.1.4 DelayStartOfReplay Callback

Can be used to coordinate when replaying begins; called once per repetition, per simulated client per mlreplay instance.

#### ≡ Syntax

```
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.

### 1.14.1.5 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

```
_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

Do not modify this callback.

### 1.14.1.6 FinIdentifySimulatedClient Callback

Used to clean up memory used by the call to IdentifySimulatedClient for the given simulated client; called once per simulated client.

#### Syntax

```
_MLREPLAY_EXPORT void _MLREPLAY_CDECL FinIdentifySimulatedClient (  
    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.

### 1.14.1.7 GetDownloadApplyTime Callback

Used to simulate slow devices; called once per download, per simulated client, and per repetition.

#### ≡ Syntax

```
_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.

### 1.14.1.8 GetMLReplayAPIVersion Callback

Used to return the replay API version.

#### ≡ Syntax

```
_MLREPLAY_EXPORT asa_uint32 _MLREPLAY_CDECL GetMLReplayAPIVersion( void )
```

## Remarks

This callback is called once per mlreplay instance and should not be modified.

### 1.14.1.9 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.

#### i 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

```
_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.

## 1.14.1.10 GlobalFini Callback

Used to clean up any global variables used by the other callbacks; called once per mlreplay instance.

### ☰ Syntax

```
_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.

## 1.14.1.11 GlobalInit Callback

Used to initialize any global variables used by the other callbacks; called once per mlreplay instance.

### ☰ Syntax

```
_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.

## 1.14.1.12 IdentifySimulatedClient Callback

Used to specify the simulated client information; called once per simulated client.

### ≡ Syntax

```
_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.

### 1.14.1.13 ReportEndOfReplay Callback

Used to perform any actions required when a simulated client is finished replaying; called once per simulated client, and per repetition.

#### ≡ Syntax

```
_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 effect 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.

## 1.14.2 MobiLink Server System Procedures

MobiLink provides the following stored procedures to help you create your applications.

### i Note

This section does not apply to SAP HANA because SQL Central does not have a MobiLink plug-in for SAP HANA.

### i Note

Support for IBM DB2 consolidated databases is deprecated.

## 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 SQL 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_`.

#### **i Note**

Support for IBM DB2 consolidated databases is deprecated.

## **System Procedures for Managing Remote Tasks**

#### **i Note**

This section does not apply to SAP HANA because SQL Central does not have a MobiLink plug-in for SAP HANA.

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
- `ml_add_ldap_server` system procedure
- `ml_add_user_auth_policy` system procedure

## Synchronization Model System Procedures

### i Note

This section does not apply to SAP HANA because SQL Central does not have a MobiLink plug-in for SAP HANA.

The following stored procedures can be used to manage schema upgrades:

- `ml_model_drop` system procedure
- `ml_model_check_all_schema` system procedure
- `ml_model_check_version_schema` system procedure

## Other System Procedures

- `ml_add_property` system procedure
- `ml_delete_sync_state_before` system procedure
- `ml_reset_sync_state` system procedure

### In this section:

[ml\\_add\\_certificates\\_file System Procedure \[page 587\]](#)

Set up trusted certificates when using TLS with LDAP authentication.

[ml\\_add\\_column System Procedure \(Deprecated\) \[page 588\]](#)

Register information about columns on remote databases for use by named column parameters.

[ml\\_add\\_connection\\_script System Procedure \[page 589\]](#)

Add or delete SQL connection scripts in the consolidated database.

[ml\\_add\\_dnet\\_connection\\_script System Procedure \[page 591\]](#)

Register or unregister a .NET method as the script for a connection event.

[ml\\_add\\_dnet\\_table\\_script System Procedure \[page 592\]](#)

Register or unregister a .NET method as the script for a table event.

[ml\\_add\\_java\\_connection\\_script System Procedure \[page 593\]](#)

Register or unregister a Java method as the script for a connection event.

[ml\\_add\\_java\\_table\\_script System Procedure \[page 595\]](#)



Register or unregister a Java method as the script for a table event.

[ml\\_add\\_lang\\_connection\\_script System Procedure \[page 597\]](#)

This procedure is for internal use only.

[ml\\_add\\_lang\\_connection\\_script\\_chk System Procedure \[page 597\]](#)

This procedure is for internal use only.

[ml\\_add\\_lang\\_table\\_script System Procedure \[page 597\]](#)

This procedure is for internal use only.

[ml\\_add\\_lang\\_table\\_script\\_chk System Procedure \[page 598\]](#)

This procedure is for internal use only.

[ml\\_add\\_ldap\\_server System Procedure \[page 598\]](#)

Create, drop or update LDAP servers.

[ml\\_add\\_missing\\_dnld\\_scripts System Procedure \[page 600\]](#)

Define missing download\_cursor and download\_delete\_cursor scripts as ignored scripts.

[ml\\_add\\_passthrough System Procedure \[page 600\]](#)

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.

[ml\\_add\\_passthrough\\_repair System Procedure \[page 601\]](#)

Define rules for handling script errors.

[ml\\_add\\_passthrough\\_script System Procedure \[page 603\]](#)

Create a passthrough script. This procedure adds an entry to the ml\_passthrough\_script system table.

[ml\\_add\\_property System Procedure \[page 605\]](#)

Add or delete MobiLink properties. This system procedure changes rows in the ml\_property system table.

[ml\\_add\\_table\\_script System Procedure \[page 609\]](#)

Add or delete SQL table scripts in the consolidated database.

[ml\\_add\\_user System Procedure \[page 611\]](#)

This procedure is for internal use only.

[ml\\_add\\_user\\_auth\\_policy System Procedure \[page 611\]](#)

Add MobiLink user authentication policies.

[ml\\_delete\\_passthrough System Procedure \(Deprecated\) \[page 613\]](#)

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.

[ml\\_delete\\_passthrough\\_repair System Procedure \(Deprecated\) \[page 614\]](#)

Delete a repair rule from the ml\_passthrough\_repair system table.

[ml\\_delete\\_passthrough\\_script System Procedure \(Deprecated\) \[page 615\]](#)

Delete a passthrough script from the ml\_passthrough\_script system table.

[ml\\_delete\\_sync\\_state System Procedure \[page 615\]](#)

Delete unused or unwanted synchronization states.

[ml\\_delete\\_sync\\_state\\_before System Procedure \[page 617\]](#)

Clean up the MobiLink system tables when you have dropped remote databases.

[ml\\_delete\\_user System Procedure \[page 618\]](#)

This procedure is for internal use only.

[ml\\_model\\_drop System Procedure \[page 618\]](#)

Drop synchronization models installed using the MobiLink 17 plug-in for SQL Central.

[ml\\_model\\_check\\_all\\_schema System Procedure \[page 619\]](#)

Check the status of each schema object required by deployed synchronization models. This stored procedure returns information for all deployed synchronization models.

[ml\\_model\\_check\\_version\\_schema System Procedure \[page 621\]](#)

Check the status of each schema object required by deployed synchronization models. This stored procedure returns information for the specified script version.

[ml\\_ra\\_add\\_agent\\_id System Procedure \[page 623\]](#)

Define a new remote agent in the consolidated database.

[ml\\_ra\\_assign\\_task System Procedure \[page 623\]](#)

Assign a task to a specific remote agent.

[ml\\_ra\\_cancel\\_notification System Procedure \[page 624\]](#)

Cancel a server initiated remote task (SIRT) request that is no longer needed.

[ml\\_ra\\_cancel\\_task\\_instance System Procedure \[page 625\]](#)

Cancel a remote task instance that is no longer needed.

[ml\\_ra\\_clone\\_agent\\_properties System Procedure \[page 626\]](#)

Set all remote agent properties at once.

[ml\\_ra\\_delete\\_agent\\_id System Procedure \[page 627\]](#)

Delete a defined agent from the consolidated database.

[ml\\_ra\\_delete\\_events\\_before System Procedure \[page 627\]](#)

Delete events that are no longer needed from the consolidated database.

[ml\\_ra\\_delete\\_remote\\_id System Procedure \[page 628\]](#)

Delete a remote database that is no longer needed from the consolidated database.

[ml\\_ra\\_delete\\_task System Procedure \[page 629\]](#)

Delete a remote task from the consolidated database.

[ml\\_ra\\_get\\_agent\\_events System Procedure \[page 629\]](#)

Query events.

[ml\\_ra\\_get\\_agent\\_ids System Procedure \[page 633\]](#)

Get all the agents in the consolidated database.

[ml\\_ra\\_get\\_agent\\_properties System Procedure \[page 634\]](#)

See all the properties set for an agent.

[ml\\_ra\\_get\\_latest\\_event\\_id System Procedure \[page 635\]](#)

Help determine how many new events there are.

[ml\\_ra\\_get\\_orphan\\_taskdbs System Procedure \[page 635\]](#)

Display a list of orphan agent databases, meaning an agent database that does not have a valid agent ID.

[ml\\_ra\\_get\\_remote\\_ids System Procedure \[page 636\]](#)

Get all the remote databases in the consolidated database, excluding the agent databases.

[ml\\_ra\\_get\\_task\\_results System Procedure \[page 637\]](#)

Get events related to a specific run of a task.

[ml\\_ra\\_get\\_task\\_status System Procedure \[page 639\]](#)

Check the status of tasks.

[ml\\_ra\\_manage\\_remote\\_db System Procedure \[page 641\]](#)

Add an agent-managed remote database.

[ml\\_ra\\_notify\\_agent\\_sync System Procedure \[page 642\]](#)

Cause an agent to synchronize its state.

[ml\\_ra\\_notify\\_task System Procedure \[page 642\]](#)

Run a task using server initiated remote tasks (SIRT).

[ml\\_ra\\_reassign\\_taskdb System Procedure \[page 643\]](#)

Reassign an agent database in the situation where you have an orphan agent database.

[ml\\_ra\\_set\\_agent\\_property System Procedure \[page 644\]](#)

Set remote agent properties.

[ml\\_ra\\_unmanage\\_remote\\_db System Procedure \[page 645\]](#)

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.

[ml\\_reset\\_sync\\_state System Procedure \[page 645\]](#)

Reset synchronization state information in MobiLink system tables.

[ml\\_server\\_delete System Procedure \[page 647\]](#)

This procedure is for internal use only.

[ml\\_server\\_update System Procedure \[page 647\]](#)

This procedure is for internal use only.

## 1.14.2.1 ml\_add\_certificates\_file System Procedure

Set up trusted certificates when using TLS with LDAP authentication.

≡ Syntax

```
ml_add_certificates_file (  
  'file_name',  
)
```

### Parameters

Syntax	Description
file_name	VARCHAR(1024). The name of the trusted certificates file.

## 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.

## Related Information

[ml\\_add\\_ldap\\_server System Procedure \[page 598\]](#)

[ml\\_add\\_user\\_auth\\_policy System Procedure \[page 611\]](#)

## 1.14.2.2 ml\_add\_column System Procedure (Deprecated)

Register information about columns on remote databases for use by named column parameters.

### ≡ Syntax

```
ml_add_column (  
  'version',  
  'table',  
  'column',  
  'type'  
)
```

## Parameters

Syntax	Description
<code>version</code>	VARCHAR(128). The version name.
<code>table</code>	VARCHAR(128). The table name.
<code>column</code>	VARCHAR(128). The column name.
<code>type</code>	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.

## Example

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 )
```

## Related Information

[Script Parameters \[page 294\]](#)

### 1.14.2.3 ml\_add\_connection\_script System Procedure

Add or delete SQL connection scripts in the consolidated database.

#### ≡ Syntax

```
ml_add_connection_script (  
  'version',  
  'event',  
  'script'
```

## Parameters

Syntax	Description
<code>version</code>	VARCHAR(128). The version name.
<code>event</code>	VARCHAR(128). The event name.
<code>script</code>	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.

## 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.

## Example

The following statement adds a connection script associated with the begin\_synchronization event to the script version custdb in a SQL Anywhere consolidated database. The script itself is the single statement that sets the @EmployeeID variable.

```
call ml_add_connection_script( 'custdb',
    'begin_synchronization',
    'set @EmployeeID = {ml s.username}' )
```

## Related Information

[Script Additions and Deletions \[page 316\]](#)

[ml\\_add\\_table\\_script System Procedure \[page 609\]](#)

[ml\\_add\\_dnet\\_connection\\_script System Procedure \[page 591\]](#)

[ml\\_add\\_dnet\\_table\\_script System Procedure \[page 592\]](#)

[ml\\_add\\_java\\_connection\\_script System Procedure \[page 593\]](#)

## 1.14.2.4 ml\_add\_dnet\_connection\_script System Procedure

Register or unregister a .NET method as the script for a connection event.

### Syntax

```
ml_add_dnet_connection_script (  
  'version',  
  'event',  
  'script'  
)
```

## Parameters

Syntax	Description
<code>version</code>	VARCHAR(128). The version name.
<code>event</code>	VARCHAR(128). The event name.
<code>script</code>	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.

## 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.

## Example

The following example registers the `beginDownloadConnection` method of the `ExampleClass` class for the `begin_download` event.

```
call ml_add_dnet_connection_script( 'ver1',
```

```
'begin_download',  
'ExamplePackage.ExampleClass.beginDownloadConnection' );
```

## Related Information

[Script Additions and Deletions \[page 316\]](#)

[.NET Methods \[page 547\]](#)

[Synchronization Scripts in Microsoft .NET \[page 541\]](#)

[ml\\_add\\_dnet\\_table\\_script System Procedure \[page 592\]](#)

[ml\\_add\\_connection\\_script System Procedure \[page 589\]](#)

[ml\\_add\\_table\\_script System Procedure \[page 609\]](#)

[ml\\_add\\_java\\_table\\_script System Procedure \[page 595\]](#)

## 1.14.2.5 ml\_add\_dnet\_table\_script System Procedure

Register or unregister a .NET method as the script for a table event.

### ≡ Syntax

```
ml_add_dnet_table_script (  
  'version',  
  'table',  
  'event',  
  'script'  
)
```

## Parameters

Syntax	Description
<i>version</i>	VARCHAR(128). The version name.
<i>table</i>	VARCHAR(128). The table name.
<i>event</i>	VARCHAR(128). The event name.
<i>script</i>	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.



## 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.

This procedure can only be used with non-data table scripts. All table row data must be handled using direct row handling, via the `handle_UploadData` and `handle_DownloadData` connection events. To register these data scripts, use the `ml_add_dnet_connection_script` procedure.

## Example

The following example assigns the `empDownloadCursor` method of the `EgClass` class to the `download_cursor` event for the table `emp`.

```
call ml_add_dnet_table_script( 'ver1', 'emp',  
'download_cursor', 'EgPackage.EgClass.empDownloadCursor' )
```

## Related Information

[Script Additions and Deletions \[page 316\]](#)

[.NET Methods \[page 547\]](#)

[Synchronization Scripts in Microsoft .NET \[page 541\]](#)

[Direct Row Handling \[page 558\]](#)

[Data Scripts \[page 346\]](#)

[ml\\_add\\_dnet\\_connection\\_script System Procedure \[page 591\]](#)

[ml\\_add\\_connection\\_script System Procedure \[page 589\]](#)

[ml\\_add\\_table\\_script System Procedure \[page 609\]](#)

[ml\\_add\\_java\\_connection\\_script System Procedure \[page 593\]](#)

[handle\\_UploadData Connection Event \[page 448\]](#)

[handle\\_DownloadData Connection Event \[page 437\]](#)

## 1.14.2.6 ml\_add\_java\_connection\_script System Procedure

Register or unregister a Java method as the script for a connection event.

≡ Syntax

```
ml_add_java_connection_script (  
'version',
```

```
'event',  
'script',  
)
```

## Parameters

Syntax	Description
<code>version</code>	VARCHAR(128). The version name.
<code>event</code>	VARCHAR(128). The event name.
<code>script</code>	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.

**i Note**  
Support for IBM DB2 consolidated databases is deprecated.

## 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.

## Example

The following example registers the `endConnection` method of the `CustEmpScripts` class for the `end_connection` event.

```
call ml_add_java_connection_script( 'ver1',  
'end_connection',  
'CustEmpScripts.endConnection' )
```

## Related Information

[Script Additions and Deletions \[page 316\]](#)

[Java Methods \[page 531\]](#)

[Synchronization Script Writing in Java \[page 526\]](#)

[ml\\_add\\_connection\\_script System Procedure \[page 589\]](#)

[ml\\_add\\_table\\_script System Procedure \[page 609\]](#)

[ml\\_add\\_dnet\\_connection\\_script System Procedure \[page 591\]](#)

[ml\\_add\\_dnet\\_table\\_script System Procedure \[page 592\]](#)

[ml\\_add\\_java\\_table\\_script System Procedure \[page 595\]](#)

### 1.14.2.7 ml\_add\_java\_table\_script System Procedure

Register or unregister a Java method as the script for a table event.

#### Syntax

```
ml_add_java_table_script (  
  'version',  
  'table',  
  'event',  
  'script'  
)
```

## Parameters

Syntax	Description
<code>version</code>	VARCHAR(128). The version name.
<code>table</code>	VARCHAR(128). The table name.
<code>event</code>	VARCHAR(128). The event name.
<code>script</code>	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.

#### i Note

Support for IBM DB2 consolidated databases is deprecated.

## 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.

This procedure can only be used with non-data table scripts. All table row data must be handled using direct row handling, via the `handle_UploadData` and `handle_DownloadData` connection events. To register these data scripts, use the `ml_add_dnet_connection_script` procedure.

## 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' )
```

## Related Information

[Script Additions and Deletions \[page 316\]](#)

[Java Methods \[page 531\]](#)

[Synchronization Script Writing in Java \[page 526\]](#)

[Direct Row Handling \[page 558\]](#)

[Data Scripts \[page 346\]](#)

[ml\\_add\\_connection\\_script System Procedure \[page 589\]](#)

[ml\\_add\\_table\\_script System Procedure \[page 609\]](#)

[ml\\_add\\_dnet\\_connection\\_script System Procedure \[page 591\]](#)

[ml\\_add\\_dnet\\_table\\_script System Procedure \[page 592\]](#)

[ml\\_add\\_java\\_connection\\_script System Procedure \[page 593\]](#)

[handle\\_UploadData Connection Event \[page 448\]](#)

[handle\\_DownloadData Connection Event \[page 437\]](#)

## 1.14.2.8 ml\_add\_lang\_connection\_script System Procedure

This procedure is for internal use only.

### Related Information

[ml\\_add\\_connection\\_script System Procedure \[page 589\]](#)

[ml\\_add\\_dnet\\_connection\\_script System Procedure \[page 591\]](#)

[ml\\_add\\_java\\_connection\\_script System Procedure \[page 593\]](#)

## 1.14.2.9 ml\_add\_lang\_connection\_script\_chk System Procedure

This procedure is for internal use only.

### Related Information

[ml\\_add\\_connection\\_script System Procedure \[page 589\]](#)

[ml\\_add\\_dnet\\_connection\\_script System Procedure \[page 591\]](#)

[ml\\_add\\_java\\_connection\\_script System Procedure \[page 593\]](#)

## 1.14.2.10 ml\_add\_lang\_table\_script System Procedure

This procedure is for internal use only.

### Related Information

[ml\\_add\\_table\\_script System Procedure \[page 609\]](#)

[ml\\_add\\_java\\_table\\_script System Procedure \[page 595\]](#)

[ml\\_add\\_dnet\\_table\\_script System Procedure \[page 592\]](#)

## 1.14.2.11 ml\_add\_lang\_table\_script\_chk System Procedure

This procedure is for internal use only.

### Related Information

[ml\\_add\\_table\\_script System Procedure \[page 609\]](#)

[ml\\_add\\_java\\_table\\_script System Procedure \[page 595\]](#)

[ml\\_add\\_dnet\\_table\\_script System Procedure \[page 592\]](#)

## 1.14.2.12 ml\_add\_ldap\_server System Procedure

Create, drop or update LDAP servers.

### ☰ Syntax

```
ml_add_ldap_server (  
  'ldsrv_name',  
  'search_url',  
  'access_dn',  
  'access_dn_pwd',  
  'auth_url',  
  'conn_retries',  
  'conn_timeout',  
  'use_tls'  
)
```

### Parameters

Syntax	Description
<code>ldsrv_name</code>	VARCHAR(128). A unique LDAP server name.
<code>search_url</code>	VARCHAR(1024). A URL string that identifies the host by name or IP address, port number, and search string to perform the DN (distinguished name) lookup for a given user id.
<code>access_dn</code>	VARCHAR(1024). The distinguished name for an LDAP user that is used by the MobiLink server to connect to the LDAP server. The LDAP user must have permission on the LDAP server to search for DNs.

Syntax	Description
<code>access_dn_pwd</code>	VARCHAR(1024). The password associated with the DN specified with the <code>access_dn</code> parameter.
<code>auth_url</code>	VARCHAR(1024). A URL string that identifies the host by name or IP address and the port number of the LDAP server used to authenticate a user.
<code>conn_retries</code>	TINYINT. The number of times the MobiLink server tries to connect to the LDAP server for DN searches and authentication. The valid range is 1-60. The default is 3.
<code>conn_timeout</code>	TINYINT. The connection timeout from the MobiLink server to the LDAP server for DN searches and authentication. The value is specified in seconds. The default value is 10 seconds.
<code>start_tls</code>	TINYINT. Specifies that TLS be used for connections to the LDAP server for DN searches and authentication.

## Remarks

This procedure populates the `ml_ldap_server` table with information about the specified LDAP server.

## Example

The following example adds an LDAP server named `my_primary` into the `ml_ldap_server` table.

```
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
)
```

## Related Information

[ml\\_add\\_certificates\\_file System Procedure \[page 587\]](#)

[ml\\_add\\_user\\_auth\\_policy System Procedure \[page 611\]](#)

## 1.14.2.13 ml\_add\_missing\_dnld\_scripts System Procedure

Define missing download\_cursor and download\_delete\_cursor scripts as ignored scripts.

### ☰ Syntax

```
ml_add_missing_dnld_scripts (  
  'script_version_name')
```

## Parameters

Syntax	Description
<code>script_version_name</code>	VARCHAR(128). The name of the script version.

## Related Information

[download\\_cursor Table Event \[page 393\]](#)

[download\\_delete\\_cursor Table Event \[page 396\]](#)

## 1.14.2.14 ml\_add\_passthrough System Procedure

Identify remote databases that should execute a script. This procedure adds an entry to the ml\_passthrough system table. If an entry with the given remote\_id and run\_order already exists in the table, this procedure updates the entry.

### ☰ Syntax

```
ml_add_passthrough (  
  'remote_id',  
  'script_name',  
  run_order  
)
```



## Parameters

Syntax	Description
<code>remote_id</code>	<p>VARCHAR(128). The remote ID of the database that should execute the script. This value can be a valid remote ID in the <code>ml_database</code> table to apply to a specific client, or null to apply to all the script clients listed in the <code>ml_database</code> table.</p> <div style="border: 1px solid #ccc; padding: 5px;"><p><b>⚠ Caution</b></p><p>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.</p></div>
<code>script_name</code>	<p>VARCHAR(128). The name of the script being subscribed to. This value must be a valid script name defined in the <code>ml_passthrough_script</code> table.</p>
<code>run_order</code>	<p>INTEGER. The <code>run_order</code> parameter determines the order in which scripts are applied on the remote database. Scripts are always applied in order by <code>run_order</code>. Each remote database stores the <code>run_order</code> of the last script that it attempted to apply and does not download or execute any script with a <code>run_order</code> less than this.</p> <p>This value must be a non-negative integer or null.</p>

## 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.

### 1.14.2.15 ml\_add\_passthrough\_repair System Procedure

Define rules for handling script errors.

Each rule defines the action that a client should perform when a specific script generates a given error code. This procedure adds an entry to the `ml_passthrough_repair` system table. If an entry with the given `failed_script_name` and `error_code` already exists in the table, the procedure updates the entry.

#### ⌘ Syntax

```
ml_add_passthrough_repair (  
  'failed_script_name',
```

```

error_code,
'new_script_name',
'action'
)

```

## Parameters

Syntax	Description
<code>failed_script_name</code>	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.
<code>error_code</code>	INTEGER. The SQL Anywhere error code that this rule handles.
<code>new_script_name</code>	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.
<code>action</code>	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: <p><b>R</b></p> <p>(replace) Indicates that the failed script should be replaced with the one specified by <i>new script name</i> and an attempt should be made to run the new script. To rerun the failed script, choose <i>new script name</i> to be the same as <i>failed script name</i>.</p> <p><b>P</b></p> <p>(purge) Indicates that the remote database should discard all the scripts that it has received and continue executing script normally after that.</p> <p><b>S</b></p> <p>(skip) Indicates that the remote database should ignore the failed script and continue executing scripts as if the failed script had succeeded.</p> <p><b>H</b></p> <p>(halt) Indicates that the remote database should not execute any more scripts until it receives further instructions.</p>

## Remarks

You should make every effort to avoid failed SQL passthrough scripts by testing scripts thoroughly.

### 1.14.2.16 ml\_add\_passthrough\_script System Procedure

Create a passthrough script. This procedure adds an entry to the ml\_passthrough\_script system table.

#### ☰ Syntax

```
ml_add_passthrough_script (  
  'script_name',  
  'flags',  
  'affected_pubs',  
  'script',  
  'description'  
)
```

## Parameters

Syntax	Description
<code>script_name</code>	VARCHAR(128). The script name. This value must be unique.

## Syntax

## Description

`flags`

VARCHAR(256). The value that tells clients how to run the script. This value can be null or contain a combination of the following keywords in a semicolon-delimited list:

### **manual**

Indicates that the script may only be run in manual execution mode. By default, all scripts can be run in either automatic or manual execution modes.

### **exclusive**

Indicates that the script may only be automatically executed at the end of a synchronization where exclusive locks were obtained on all tables being synchronized. This option is ignored if the `affected_pubs` value lists no publications. This option is only meaningful to SQL Anywhere remotes.

### **schema\_diff**

Indicates that the script should be run in schema-diffing mode. In this mode, the database schema is altered to match the schema described in the script. For example, a create statement for an existing table is treated as an alter statement. This flag only applies to scripts run on UltraLite remotes.

For example:

```
'manual;exclusive;schema_diff'
```

`affected_pubs`

TEXT. A list of publications that must be synchronized before the script is run. An empty string or null indicates that no synchronization is required. This value is only meaningful for SQL Anywhere clients. For Adaptive Server Enterprise, this parameter is VARCHAR(16384). For IBM DB2 LUW, this parameter is VARCHAR(4000). For Oracle, this parameter is CLOB.

Syntax	Description
<code>script</code>	<p>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.</p> <p>The <code>script</code> content must be non-null. For UltraLite remotes, the <code>script</code> 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 <code>script</code> content can be any collection of SQL statements that are valid when enclosed by a <code>begin...end</code> block.</p> <p>Example of <code>script</code> content on a SQL Anywhere remote:</p> <pre>DECLARE val INTEGER; SELECT c1 INTO val FROM t1 WHERE pk = 5; IF val &gt; 100 THEN     INSERT INTO t2 VALUES ('c1 is big'); ENDIF</pre> <p>Example of <code>script</code> content on an UltraLite remote:</p> <pre>CREATE TABLE myScript (c1 INT NOT NULL PRIMARY KEY) GO INSERT INTO myScript VALUES (1) GO</pre>
<code>description</code>	<p>VARCHAR(2000). A comment or description of the script. This value may be null.</p>

## Remarks

This procedure generates an error if the specified `script_name` already exists in `ml_passthrough_script`.

### 1.14.2.17 ml\_add\_property System Procedure

Add or delete MobiLink properties. This system procedure changes rows in the `ml_property` system table.

#### ☰ Syntax

```
ml_add_property (
    'comp_name',
    'prop_set_name',
    'prop_name',
    'prop_value'
)
```

## Parameters

Syntax	Description
<code>comp_name</code>	<p>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.</p>
<code>prop_set_name</code>	<p>VARCHAR(128). The property set name.</p> <p>If the component name is ScriptVersion, then this parameter is the name of the script version.</p> <p>If the component name is MLS, then this parameter can be <code>ml_user_log_verbosity</code> to specify verbosity for a MobiLink user, <code>ml_remote_id_log_verbosity</code> to specify verbosity for a remote ID, or <code>locking_and_blocking_detection</code> to report locking and blocking information to the MobiLink Profiler or MobiLink server log file.</p> <p>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.</p>
<code>prop_name</code>	<p>VARCHAR(128). The property name.</p> <p>If the component name is ScriptVersion, then this parameter is a property that you define. You can reference these properties using <code>DBConnectionContext: getVersion</code> and <code>getProperties</code>, or <code>ServerContext: getPropertiesByVersion</code>, <code>getProperties</code>, and <code>getPropertySetNames</code>.</p> <p>If the component name is MLS, then this property is either a MobiLink user name or remote ID that you define, or <code>blocking_threshold_in_seconds</code> for reporting locking and blocking information to the MobiLink Profiler or MobiLink server log file.</p>

Syntax	Description
<code>prop_value</code>	<p>TEXT. The property value.</p> <p>If the <code>prop_set_name</code> is <code>ml_user_log_verbosity</code> or <code>ml_remote_id_log_verbosity</code>, this must be a valid <code>mlsrv -v</code> option.</p> <p>For reporting locking and blocking information to the MobiLink Profiler or MobiLink server log file, this value is <code>time_in_seconds</code>.</p> <p>For Adaptive Server Enterprise, this parameter is <code>VARCHAR(16384)</code>. For IBM DB2 LUW, this parameter is <code>VARCHAR(4000)</code>. For Oracle, this parameter is <code>CLOB</code>. To delete a property, set to null.</p>

**i Note**

Support for IBM DB2 consolidated databases is deprecated.

## 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:

```
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:

```
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:

```
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.

To disable log verbosity for a MobiLink user, log into the consolidated database and execute the following SQL statement:

```
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:

```
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, 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' );
```

## Related Information

[MobiLink Server Settings for Server-initiated Synchronization](#)  
[-v mlsrv17 Option \[page 92\]](#)

### 1.14.2.18 ml\_add\_table\_script System Procedure

Add or delete SQL table scripts in the consolidated database.

#### Syntax

```
ml_add_table_script (  
  'version',  
  'table',  
  'event',  
  'script'  
)
```

## Parameters

Syntax	Description
<code>version</code>	VARCHAR(128). The version name.
<code>table</code>	VARCHAR(128). The table name.
<code>event</code>	VARCHAR(128). The event name.

Syntax	Description
<code>script</code>	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.

**i Note**  
Support for IBM DB2 consolidated databases is deprecated.

## 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 mlsrv17 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.

## Example

The following command adds a table script associated with the upload\_insert event on the Customer table.

```
call ml_add_table_script( 'default', 'Customer', 'upload_insert',
  'INSERT INTO Customer( cust_id, name, rep_id, active )
  VALUES ( {ml r.cust_id}, {ml r.name}, {ml r.rep_id}, 1 )' )
```

## Related Information

[Script Additions and Deletions \[page 316\]](#)

[ml\\_add\\_connection\\_script System Procedure \[page 589\]](#)

[ml\\_add\\_dnet\\_connection\\_script System Procedure \[page 591\]](#)

[ml\\_add\\_dnet\\_table\\_script System Procedure \[page 592\]](#)

[ml\\_add\\_java\\_connection\\_script System Procedure \[page 593\]](#)

[ml\\_add\\_java\\_table\\_script System Procedure \[page 595\]](#)

[-zf mlsrv17 Option \[page 107\]](#)

## 1.14.2.19 ml\_add\_user System Procedure

This procedure is for internal use only.

## 1.14.2.20 ml\_add\_user\_auth\_policy System Procedure

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

Syntax	Description
<code>policy_name</code>	VARCHAR(128). A unique user authentication policy name.
<code>primary_ldsrv_name</code>	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.
<code>secondary_ldsrv_name</code>	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.

Syntax	Description
<code>ldap_auto_failback_period</code>	<p>INTEGER. Use this parameter to inform the MobiLink server when it should fail over to the primary LDAP server for user authentication. The time is specified in seconds and the default value is 900 seconds (15 minutes).</p> <p>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 switches back to use the primary server for user authentication for any users who are currently using this user authentication policy when the elapsed time since the failure was detected has reached <code>@ldap_auto_failback_period</code>.</p>
<code>ldap_failover_to_std</code>	<p>INTEGER. Specifies whether the MobiLink server should use standard methods (password and user authentication scripts) to authenticate the user. The value can be as follows:</p> <ul style="list-style-type: none"> <li><b>0</b> The MobiLink server authenticates the user only against LDAP servers. If the user cannot be authenticated against an LDAP server, then the synchronization request fails.</li> <li><b>1</b> The MobiLink server authenticates the user by using the script-based method of user authentication if an LDAP server is not available.</li> <li><b>2</b> The MobiLink server authenticates the user against an LDAP server first and then authenticates the user with the script-based method of user authentication, whether or not the user is authenticated with the LDAP server. The MobiLink server passes one of the following values to indicate the user authentication status to the scripts: 1000 if the user is authenticated against the LDAP server; 4000 if the user is not authenticated against the LDAP server; or 6000 if the LDAP servers are not available.</li> </ul> <p>The MobiLink user password is only hashed and stored in the <code>ml_user</code> table in the consolidated database if the <code>ldap_failover_to_std</code> parameter is configured with a value of 1 or 2. The password is not saved if this parameter is set to 0.</p>

## Remarks

If the specified `policy_name` does not exist in the table, then this procedure adds a user authentication policy to the `ml_user_auth_policy` table. If the `policy_name` is already in the table, then 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 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.

## Related Information

[ml\\_add\\_ldap\\_server System Procedure \[page 598\]](#)

[ml\\_add\\_certificates\\_file System Procedure \[page 587\]](#)

### 1.14.2.21 ml\_delete\_passthrough System Procedure (Deprecated)

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

Syntax	Description
<code>remote_id</code>	VARCHAR(128). The remote ID. If <code>remote_id</code> is null then all rows in the <code>ml_passthrough</code> table for the specified script name and run order are removed.
<code>script_name</code>	VARCHAR(128). The script name.

Syntax	Description
<code>run_order</code>	INTEGER. The run order of the script applied on the remote database. If <code>run_order</code> is null then all rows for the specified <code>remote_id</code> and <code>script_name</code> are removed from the <code>ml_passthrough</code> table regardless of their run order.

## Remarks

The MobiLink server does not automatically remove entries from the `ml_passthrough` table. You must use this procedure to remove outdated passthrough scripts.

## 1.14.2.22 ml\_delete\_passthrough\_repair System Procedure (Deprecated)

Delete a repair rule from the `ml_passthrough_repair` system table.

☰ Syntax

```
ml_delete_passthrough_repair (
  'failed_script_name',
  error_code
)
```

## Parameters

Syntax	Description
<code>failed_script_name</code>	VARCHAR(128). The name of the script to which a rule applied.
<code>error_code</code>	INTEGER. The error code for which the rule applied.

## Remarks

The MobiLink server does not automatically remove entries from the `ml_passthrough_repair` table. You must use this procedure to remove outdated passthrough repair scripts.

## 1.14.2.23 ml\_delete\_passthrough\_script System Procedure (Deprecated)

Delete a passthrough script from the ml\_passthrough\_script system table.

### ≡ Syntax

```
ml_delete_passthrough_script (  
  'script_name'  
)
```

## Parameters

Syntax	Description
<code>script_name</code>	VARCHAR(128). The name of the script to remove.

## 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.

## 1.14.2.24 ml\_delete\_sync\_state System Procedure

Delete unused or unwanted synchronization states.

### ≡ Syntax

```
ml_delete_sync_state (  
  'user',  
  'remote_id'  
)
```

## Parameters

Syntax	Description
<code>user</code>	VARCHAR(128). The MobiLink user name.
<code>remote_id</code>	VARCHAR(128). The remote ID.

## Remarks

These parameters can be null. If all the parameters are null, the procedure does nothing.

This stored procedure deletes all the rows from the `ml_subscription` table for the given MobiLink user name and remote ID. It also removes this remote ID from the `ml_database` table, if the remote ID is no longer referenced by any rows in the `ml_subscription` table.

If the remote ID is null and the MobiLink user name is not null, it removes all the rows that are referenced by the given MobiLink user name from the `ml_subscription` table and all the remote IDs from the `ml_database` table, if these remote IDs are no longer referenced by any rows in the `ml_subscription` table.

If the MobiLink user name is null and the remote ID is not null, the MobiLink user is not removed by this stored procedure, even if all the remote IDs have been deleted from the `ml_database` table and this user is no longer referenced by any rows in the `ml_subscription` table. If this MobiLink user needs to be deleted, you may delete it by issuing a command such as

```
delete from ml_user where name = 'user_name'
```

where `user_name` is the MobiLink user you want to delete.

Use this stored procedure with extreme caution because the MobiLink server automatically adds this remote ID in the `ml_database` and `ml_subscription` tables without checking its synchronization status the next time the MobiLink client requests synchronization for this remote ID. It may cause data inconsistency to delete synchronization states for a remote ID that did not have a successful synchronization in the last synchronization attempt.

This procedure removes all the rows from the `ml_subscription` table and the `ml_database` table for the given remote ID.

## Example

The following example cleans up MobiLink system table information about remote databases with the remote ID `remote_db_for_John` for the MobiLink user John:

```
CALL ml_delete_sync_state( 'John', 'remote_db_for_John' )
```



## 1.14.2.25 ml\_delete\_sync\_state\_before System Procedure

Clean up the MobiLink system tables when you have dropped remote databases.

### Syntax

```
ml_delete_sync_state_before (  
    'ts'  
)
```

## Parameters

Syntax	Description
ts	TIMESTAMP. The datetime must appear in exactly the order specified in the consolidated database. If the datetime format in the consolidated database is set to 'yyyy/mm/dd hh:mm:ss.ssss', then the timestamp must appear in the order year, month, day, hour, minute, second, fraction of second.

## Remarks

This stored procedure removes rows from MobiLink system tables that pertain to remote databases that are no longer being used. In particular, it does the following:

- Deletes all the rows from the ml\_subscription system table that have both the last\_upload\_time and last\_download\_time earlier than the given timestamp.
- Removes remote IDs from the ml\_database system table if the remote IDs are no longer referenced by any rows in the ml\_subscription table.

You should not use this system procedure for a time period that is so recent that it may delete rows for remote databases that have not actually been deleted. If you do, the deletion of the rows in ml\_subscription and ml\_database could cause problems for remote databases that are in an "unknown state" caused by an unsuccessful upload; in that unknown state, the remote 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' )
```

## 1.14.2.26 ml\_delete\_user System Procedure

This procedure is for internal use only.

## 1.14.2.27 ml\_model\_drop System Procedure

Drop synchronization models installed using the MobiLink 17 plug-in for SQL Central.

☰ Syntax

```
ml_model_drop (  
  'script_version'  
)
```

## Parameters

Syntax	Description
<code>script_version</code>	VARCHAR(128). The name of the script version associated with the synchronization model you want to drop.

## 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 17 plug-in.

## Related Information

[ml\\_model\\_check\\_all\\_schema System Procedure \[page 619\]](#)

[ml\\_model\\_check\\_version\\_schema System Procedure \[page 621\]](#)

### 1.14.2.28 ml\_model\_check\_all\_schema System Procedure

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 SQL 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.

## Related Information

[ml\\_model\\_drop System Procedure \[page 618\]](#)

[ml\\_model\\_check\\_version\\_schema System Procedure \[page 621\]](#)

## 1.14.2.29 ml\_model\_check\_version\_schema System Procedure

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

Syntax	Description
<code>script_version</code>	VARCHAR(128). The name of the script version associated with the synchronization model you want to check.

## 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 SQL 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.

## Related Information

[ml\\_model\\_drop System Procedure \[page 618\]](#)

[ml\\_model\\_check\\_all\\_schema System Procedure \[page 619\]](#)

## 1.14.2.30 ml\_ra\_add\_agent\_id System Procedure

Define a new remote agent in the consolidated database.

### Parameters

Syntax	Description
<code>agent_id</code>	VARCHAR(128). This is an IN parameter that specifies the ID of the new agent to be defined in the consolidated database.

### Remarks

If an Agent connects to the MobiLink server without `ml_ra_add_agent_id` being called first, then that Agent is automatically added to the consolidated database. However, all properties for that agent are default values.

### Related Information

[ml\\_ra\\_manage\\_remote\\_db System Procedure \[page 641\]](#)

[ml\\_ra\\_clone\\_agent\\_properties System Procedure \[page 626\]](#)

[ml\\_ra\\_set\\_agent\\_property System Procedure \[page 644\]](#)

## 1.14.2.31 ml\_ra\_assign\_task System Procedure

Assign a task to a specific remote agent.

### Parameters

Syntax	Description
<code>agent_id</code>	VARCHAR(128). This is an IN parameter that specifies the ID of the agent to assign the task to.
<code>task_name</code>	VARCHAR(128). This is an IN parameter that specifies the name of the specific task being assigned.

## Remarks

Tasks must first be defined in SQL Central using the MobiLink 17 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 SQL Central before calling the `task_name` parameter.

## Related Information

[ml\\_ra\\_cancel\\_task\\_instance System Procedure \[page 625\]](#)

[ml\\_ra\\_notify\\_task System Procedure \[page 642\]](#)

[ml\\_ra\\_cancel\\_notification System Procedure \[page 624\]](#)

## 1.14.2.32 ml\_ra\_cancel\_notification System Procedure

Cancel a server initiated remote task (SIRT) request that is no longer needed.

## Parameters

Syntax	Description
<code>agent_id</code>	VARCHAR(128). This is an IN parameter that specifies the ID of the agent responsible for the task you are canceling.
<code>task_name</code>	VARCHAR(128). This is an IN parameter that specifies the name of the specific task being canceled.

## Related Information

[ml\\_ra\\_notify\\_task System Procedure \[page 642\]](#)

[ml\\_ra\\_assign\\_task System Procedure \[page 623\]](#)

[ml\\_ra\\_delete\\_task System Procedure \[page 629\]](#)

[ml\\_ra\\_cancel\\_task\\_instance System Procedure \[page 625\]](#)



## 1.14.2.33 ml\_ra\_cancel\_task\_instance System Procedure

Cancel a remote task instance that is no longer needed.

### Parameters

Syntax	Description
<code>agent_id</code>	VARCHAR(128). This is an IN parameter that specifies the ID of the agent responsible for the task you are canceling.
<code>task_name</code>	VARCHAR(128). This is an IN parameter that specifies the name of the specific task being canceled.

### Remarks

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.

### Related Information

[ml\\_ra\\_cancel\\_notification System Procedure \[page 624\]](#)

[ml\\_ra\\_notify\\_task System Procedure \[page 642\]](#)

[ml\\_ra\\_assign\\_task System Procedure \[page 623\]](#)

[ml\\_ra\\_delete\\_task System Procedure \[page 629\]](#)

## 1.14.2.34 ml\_ra\_clone\_agent\_properties System Procedure

Set all remote agent properties at once.

### Parameters

Syntax	Description
<code>dst_agent_id</code>	VARCHAR(128). This is an IN parameter that specifies the ID of the destination agent being created.
<code>src_agent_id</code>	VARCHAR(128). This is an IN parameter that specifies the ID of the agent being cloned to create the new agent.

### Remarks

All the properties of an existing agent are copied to the new agent. Individual agent properties can be set more easily using SQL Central.

Assigned tasks and managed remotes are not copied to the new agent.

### Related Information

[ml\\_ra\\_set\\_agent\\_property System Procedure \[page 644\]](#)

[ml\\_ra\\_add\\_agent\\_id System Procedure \[page 623\]](#)

[ml\\_ra\\_manage\\_remote\\_db System Procedure \[page 641\]](#)

## 1.14.2.35 ml\_ra\_delete\_agent\_id System Procedure

Delete a defined agent from the consolidated database.

### Parameters

Syntax	Description
<code>agent_id</code>	VARCHAR(128). This is an IN parameter that specifies the ID of the agent being deleted.

### Remarks

If you delete an agent that was managing remote databases, those remote databases will become unmanaged.

### Related Information

[ml\\_ra\\_delete\\_events\\_before System Procedure \[page 627\]](#)

[ml\\_ra\\_delete\\_remote\\_id System Procedure \[page 628\]](#)

[ml\\_ra\\_unmanage\\_remote\\_db System Procedure \[page 645\]](#)

## 1.14.2.36 ml\_ra\_delete\_events\_before System Procedure

Delete events that are no longer needed from the consolidated database.

### Parameters

Syntax	Description
<code>delete_rows_older_than</code>	TIMESTAMP. This is an IN parameter that specifies Events older than the specified value are deleted from the consolidated database.

## Remarks

If your remote tasks return status frequently, then large numbers of events can build up in the consolidated database.

## Related Information

[ml\\_ra\\_get\\_latest\\_event\\_id System Procedure \[page 635\]](#)

## 1.14.2.37 ml\_ra\_delete\_remote\_id System Procedure

Delete a remote database that is no longer needed from the consolidated database.

## Parameters

Syntax	Description
<code>remote_id</code>	VARCHAR(128). This is an IN parameter that specifies the remote ID that corresponds to the remote database to be deleted.

## Remarks

This procedure fails if tasks are still active for the specified `remote_id`. To force deletion, first delete the `agent_id` that is managing the remote.

## Related Information

[ml\\_ra\\_delete\\_agent\\_id System Procedure \[page 627\]](#)

[ml\\_ra\\_delete\\_task System Procedure \[page 629\]](#)

## 1.14.2.38 ml\_ra\_delete\_task System Procedure

Delete a remote task from the consolidated database.

### Parameters

Syntax	Description
<code>task_name</code>	VARCHAR(128). This is an IN parameter that specifies the name of the remote task to be deleted.

### Remarks

This system procedure fails if there are still active task instances.

### Related Information

[ml\\_ra\\_delete\\_agent\\_id System Procedure \[page 627\]](#)

[ml\\_ra\\_delete\\_events\\_before System Procedure \[page 627\]](#)

[ml\\_ra\\_delete\\_remote\\_id System Procedure \[page 628\]](#)

## 1.14.2.39 ml\_ra\_get\_agent\_events System Procedure

Query events.

### Parameters

Syntax	Description
<code>start_at_event_id</code>	BIGINT. This is an IN parameter that specifies the ID of the event from which to start the query.
<code>max_events_to_fetch</code>	BIGINT. This is an IN parameter that specifies the maximum number of events to fetch.

# Returns

Result	Description
<i>event_id</i>	BIGINT. A unique ID assigned to each event. The value is incremented by 1 for each new event.
<i>event_class</i>	VARCHAR(1). The event class. The class can be either <i>I</i> for information or <i>E</i> for error.

Result	Description
<i>event_type</i>	<p>VARCHAR(8). The event types are listed below.</p> <p><b>ANEW</b></p> <p>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.</p> <p><b>AFIRST</b></p> <p>Occurs on the first synchronization of an agent.</p> <p><b>ADUP</b></p> <p>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.</p> <p><b>ARESET</b></p> <p>An agent rebuilt its agent database. Some task progress and results may have been lost.</p> <p><b>TB</b></p> <p>A task has begun execution.</p> <p><b>TE</b></p> <p>The task execution ended without a fatal error.</p> <p><b>TW</b></p> <p>A task execution is waiting for a retry interval before continuing.</p> <p><b>TAC</b></p> <p>Task execution ended because a command aborted.</p> <p><b>TAT</b></p> <p>Task execution ended because it exceeded the maximum allowed running time.</p> <p><b>TAR</b></p> <p>Task execution ended because it exceeded the maximum retry count.</p> <p><b>TFS</b></p> <p>The task completed and will not run again because it was a run-once task that succeeded.</p> <p><b>TFF</b></p> <p>The task completed and will not run again because it was a run-once task that failed.</p> <p><b>TFE</b></p> <p>The task completed and will not run again because the schedule for the task has expired.</p>

Result	Description
	<p><b>TFC</b></p> <p>The task completed and will not run again because the task was canceled by the server.</p> <p><b>CR - Command Result</b></p> <p>The result_code and result_text are populated with values specific to the type of command.</p> <p><b>CE - Command Error</b></p> <p>The result_code and result_text are populated with values specific to the type of command.</p>
<i>agent_id</i>	VARCHAR(128). The ID of the agent that produced this event.
<i>remote_id</i>	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.
<i>task_name</i>	VARCHAR(128). The name of the task. This is only set for task-related events.
<i>command_number</i>	INTEGER. The command number within a task that this event applies to. This is only set for command-specific events.
<i>run_number</i>	BIGINT. The unique number assigned to each run of a task. This is only set for task-specific events.
<i>duration</i>	INTEGER. The amount of time taken by the event. This is only set for command-specific events.
<i>event_time</i>	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.
<i>event_received</i>	TIMESTAMP. The time the event was received by the server. This is always set from the clock of the consolidated database.
<i>result_code</i>	BIGINT. An event-specific BIGINT. For example, for a SQL query command result, the code would be the SQLCODE.
<i>result_text</i>	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.
<i>p_crsr</i>	SYS_REF_CURSOR. This is an OUT parameter for Oracle only.



## Remarks

Alternatively you can use the `ml_ra_get_task_results` procedure, which only fetches events related to a specific run of a task. You can pass in a null `@run_number` to get the latest run of a task.

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.

## Related Information

[ml\\_ra\\_get\\_task\\_results System Procedure \[page 637\]](#)

[ml\\_ra\\_get\\_task\\_results System Procedure \[page 637\]](#)

## 1.14.2.40 ml\_ra\_get\_agent\_ids System Procedure

Get all the agents in the consolidated database.

## Returns

Result	Description
<i>agent_id</i>	VARCHAR(128). The agent ID.
<i>remote_id</i>	VARCHAR(128). The remote ID of the agent database.
<i>last_download_time</i>	TIMESTAMP. The last download time.
<i>last_upload_time</i>	TIMESTAMP. The last upload time.
<i>active_task_count</i>	INTEGER. The number of active tasks.
<i>description</i>	VARCHAR(2048). Reserved for future use.
<i>p_crsr</i>	SYS_REF_CURSOR. This is an OUT parameter for Oracle only.

## Related Information

[ml\\_ra\\_get\\_remote\\_ids System Procedure \[page 636\]](#)

## 1.14.2.41 ml\_ra\_get\_agent\_properties System Procedure

See all the properties set for an agent.

### Parameters

Syntax	Description
<code>agent_id</code>	VARCHAR(128). This is an IN parameter that specifies the ID of the agent you are setting properties for.

### Returns

Results	Description
<code>property_name</code>	VARCHAR(128). The property name.
<code>property_value</code>	VARCHAR(2048). The value of the property.
<code>last_modified</code>	TIMESTAMP. The time the property was last modified.
<code>p_crsr</code>	SYS_REF_CURSOR. This is an OUT parameter for Oracle only.

### Related Information

[ml\\_ra\\_get\\_agent\\_ids System Procedure \[page 633\]](#)

[ml\\_ra\\_clone\\_agent\\_properties System Procedure \[page 626\]](#)

## 1.14.2.42 ml\_ra\_get\_latest\_event\_id System Procedure

Help determine how many new events there are.

### Parameters

Syntax	Description
event_id	BIGINT. This is an OUT parameter that specifies the ID of the latest event.

### Remarks

To determine how many new events there are, call the ml\_ra\_get\_latest\_event\_id system procedure and subtract the last event\_id you processed.

### Related Information

[ml\\_ra\\_get\\_agent\\_events System Procedure \[page 629\]](#)

## 1.14.2.43 ml\_ra\_get\_orphan\_taskdbs System Procedure

Display a list of orphan agent databases, meaning an agent database that does not have a valid agent ID.

### Returns

Results	Description
<i>remote_id</i>	VARCHAR(128). The remote ID.
<i>orig_agent_id</i>	VARCHAR(128). The ID of the original agent the agent database was associated with.
<i>last_sync</i>	TIMESTAMP. The time of the last synchronization.

Results	Description
<i>p_crsr</i>	SYS_REF_CURSOR. This is an OUT parameter for Oracle only.

## Remarks

Orphaned databases can be the result of a number of problems in a synchronization system, such as creating duplicate agent ids on different computers or having two agent databases trying to use the same agent id.

The `remote_id` field has the computer name in it to aid in diagnosing problems.

## Related Information

[ml\\_ra\\_reassign\\_taskdb System Procedure \[page 643\]](#)

## 1.14.2.44 ml\_ra\_get\_remote\_ids System Procedure

Get all the remote databases in the consolidated database, excluding the agent databases.

## Parameters

None.

## Returns

Results	Description
<i>remote_id</i>	VARCHAR(128). The remote ID of the remote database.
<i>schema_name</i>	VARCHAR(128). The type of the remote database.
<i>agent_id</i>	VARCHAR(128). The agent ID of the remote database.
<i>agent_conn_str</i>	VARCHAR(2048). The agent connection string.
<i>last_download_time</i>	TIMESTAMP. The last download time.

Results	Description
<i>last_upload_time</i>	TIMESTAMP. The last upload time.
<i>description</i>	VARCHAR(128). The description of the database.
<i>p_crsr</i>	SYS_REF_CURSOR. This is an OUT parameter for Oracle only.

## Related Information

[ml\\_ra\\_get\\_agent\\_properties System Procedure \[page 634\]](#)

### 1.14.2.45 ml\_ra\_get\_task\_results System Procedure

Get events related to a specific run of a task.

## Parameters

Syntax	Description
<i>agent_id</i>	VARCHAR(128). This is an IN parameter that specifies the ID of the agent you want to get results for.
<i>task_name</i>	VARCHAR(128). This is an IN parameter that specifies the name of the task you want to get results for.
<i>run_number</i>	INTEGER. This is an IN parameter that specifies the run number you want results for.

## Returns

Result	Description
<i>event_id</i>	BIGINT. A unique ID assigned to each event. The value is incremented by 1 for each new event.
<i>event_class</i>	VARCHAR(1). The event class. The class can be either <i>I</i> for information or <i>E</i> for error.

<b>Result</b>	<b>Description</b>
<i>event_type</i>	VARCHAR(8). The event type.
<i>agent_id</i>	VARCHAR(128). The ID of the agent that produced this event.
<i>remote_id</i>	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.
<i>task_name</i>	VARCHAR(128). The name of the task. This is only set for task-related events.
<i>command_number</i>	INTEGER. The command number within a task that this event applies to. This is only set for command-specific events.
<i>run_number</i>	BIGINT. The unique number assigned to each run of a task. This is only set for task-specific events.
<i>duration</i>	INTEGER. The amount of time taken by the event. This is only set for command-specific events.
<i>event_time</i>	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.
<i>event_received</i>	TIMESTAMP. The time the event was received by the server. This is always set from the clock of the consolidated database.
<i>result_code</i>	BIGINT. An event-specific BIGINT. For example, for a SQL query command result, the code would be the SQLCODE.
<i>result_text</i>	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.
<i>p_crsr</i>	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.

## 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.

## Related Information

[ml\\_ra\\_get\\_agent\\_events System Procedure \[page 629\]](#)

## 1.14.2.46 ml\_ra\_get\_task\_status System Procedure

Check the status of tasks.

## Parameters

Syntax	Description
<code>agent_id</code>	VARCHAR(128). This is an IN parameter that specifies the ID of the agent you want to get status for.
<code>task_name</code>	VARCHAR(128). This is an IN parameter that specifies the name of the task you want to get status for.

## Returns

Result	Description
<code>agent_id</code>	VARCHAR(128). The ID of the agent that produced this event.
<code>remote_id</code>	VARCHAR(128). The ID of the remote database that the event applies to.
<code>task_name</code>	VARCHAR(128). The name of the task.
<code>task_id</code>	BIGINT. The task ID.

Result	Description
<code>state</code>	<p>VARCHAR(4). The state of the deployed task. State can be one of the following:</p> <p><b>P</b> Pending. Awaiting confirmation that the agent has received the task.</p> <p><b>A</b> Active. The agent has the task and will run it when it is scheduled to run.</p> <p><b>S</b> Succeeded. The task is complete and will not run again unless it is reassigned.</p> <p><b>F</b> Failed. The task is complete and will not run again unless it is reassigned.</p> <p><b>CP</b> Cancel pending. Waiting for confirmation that the agent has canceled the task.</p> <p><b>C</b> Canceled. The task is complete and will not run again unless it is reassigned.</p> <p><b>E</b> Expired. The task is complete and will not run again unless it is reassigned.</p>
<code>reported_exec_count</code>	BIGINT. The reported number of tasks that have been executed.
<code>reported_error_count</code>	BIGINT. The reported number of errors.
<code>reported_attempt_count</code>	BIGINT. The reported number of attempts to execute a task.
<code>last_status_update</code>	TIMESTAMP. The time the last status update was given.
<code>last_success</code>	TIMESTAMP. The time of the last successful task.
<code>assignment_time</code>	TIMESTAMP. The time the task was assigned.
<code>p_crsr</code>	SYS_REF_CURSOR. This is an OUT parameter for Oracle only.

## 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.

## Related Information

[ml\\_ra\\_get\\_task\\_results System Procedure \[page 637\]](#)

### 1.14.2.47 ml\_ra\_manage\_remote\_db System Procedure

Add an agent-managed remote database.

## Parameters

Syntax	Description
<code>agent_id</code>	VARCHAR(128). This is an IN parameter that specifies the ID of the new agent to be defined in the consolidated database.
<code>schema_name</code>	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 SQL Central.
<code>conn_str</code>	VARCHAR(128). This is an IN parameter that specifies the database connection string used by the agent to connect to the remote database.

## Related Information

[ml\\_ra\\_add\\_agent\\_id System Procedure \[page 623\]](#)

[ml\\_ra\\_clone\\_agent\\_properties System Procedure \[page 626\]](#)

## 1.14.2.48 ml\_ra\_notify\_agent\_sync System Procedure

Cause an agent to synchronize its state.

### Parameters

Syntax	Description
<code>agent_id</code>	VARCHAR(128). This is an IN parameter that specifies the ID of the agent you want to synchronize.

### Remarks

This system procedure sends new tasks to the specified agent, and causes the agent to send any results it has from the execution of tasks to the MobiLink server.

### Related Information

[ml\\_ra\\_get\\_task\\_results System Procedure \[page 637\]](#)

## 1.14.2.49 ml\_ra\_notify\_task System Procedure

Run a task using server initiated remote tasks (SIRT).

### Parameters

Syntax	Description
<code>agent_id</code>	VARCHAR(128). This is an IN parameter that specifies the ID of the agent you want to run the task.
<code>task_name</code>	VARCHAR(128). This is an IN parameter that specifies the name of the task you want the agent to execute.

## Related Information

[ml\\_ra\\_cancel\\_notification System Procedure \[page 624\]](#)

[ml\\_ra\\_delete\\_task System Procedure \[page 629\]](#)

## 1.14.2.50 ml\_ra\_reassign\_taskdb System Procedure

Reassign an agent database in the situation where you have an orphan agent database.

### Parameters

Syntax	Description
<code>taskdb_remote_id</code>	VARCHAR(128). This is an IN parameter that specifies the remote ID of the orphaned agent database.
<code>new_agent_id</code>	VARCHAR(128). This is an IN parameter that specifies the ID of the new agent you want to assign the orphaned agent database to.

### Remarks

If there are two agent databases that both want to use the same `agent_id`, the system considers the first agent database as the valid one, and the second agent database is considered an orphan, meaning it does not have a valid `agent_id` associated with it.

## Related Information

[ml\\_ra\\_get\\_orphan\\_taskdbs System Procedure \[page 635\]](#)

## 1.14.2.51 ml\_ra\_set\_agent\_property System Procedure

Set remote agent properties.

### Parameters

Syntax	Description
<code>agent_id</code>	VARCHAR(128). This is an IN parameter that specifies the agent ID.
<code>property_name</code>	VARCHAR(128). This is an IN parameter that specifies the property name to be set.
<code>property_value</code>	VARCHAR(2048). This is an IN parameter that specifies the property value to be set.

### Remarks

The agent supports the following properties:

#### **mlstream**

The MobiLink stream parameters, for example `tcpip (host=localhost)`.

#### **max\_taskdb\_sync\_interval**

The longest time in seconds that the agent should wait between synchronizing its agent database.

#### **lwp\_freq**

The time between lightweight polls.

### Related Information

[ml\\_ra\\_clone\\_agent\\_properties System Procedure \[page 626\]](#)

## 1.14.2.52 ml\_ra\_unmanage\_remote\_db System Procedure

Keep a remote database defined, but sever the link between the remote database and a remote agent, so that the database is no longer managed by its agent.

### Parameters

Syntax	Description
<code>remote_id</code>	VARCHAR(128). This is an IN parameter that specifies the remote ID to be severed.
<code>schema_name</code>	VARCHAR(128). This is an IN parameter that indicates the type of the remote database.

### Remarks

This procedure fails if there are tasks assigned to the remote database.

If you want the remote database to be managed by a different agent, you can call the `ml_ra_manage_remote_db` procedure again with a new `agent_id`.

### Related Information

[ml\\_ra\\_manage\\_remote\\_db System Procedure \[page 641\]](#)

## 1.14.2.53 ml\_reset\_sync\_state System Procedure

Reset synchronization state information in MobiLink system tables.

#### ☰ Syntax

```
ml_reset_sync_state (  
  'user',  
  'remote_id'  
)
```

## Parameters

Syntax	Description
<code>user</code>	VARCHAR(128). The MobiLink user name.
<code>remote_id</code>	VARCHAR(128). The remote ID.

## Remarks

The parameters can be null. If both parameters are null, this procedure does nothing.

This stored procedure sets the `progress`, `last_upload_time`, and `last_download_time` columns in the `ml_subscription` table to their default values for the given `user_name` and remote ID. The default value for the `progress` is 0. The default value for the `last_upload_time` and `last_download_time` columns is '1900/01/01 00:00:00'.

If the remote ID is null and the MobiLink user name is not null, this procedure sets those columns to the default values for the rows in the `ml_subscription` table referenced by the given MobiLink user name. If the MobiLink user name is null and the remote ID is not null, it sets them to the default values for the rows in the `ml_subscription` table with the given remote ID.

Use this stored procedure with extreme caution. The MobiLink server does not do any synchronization status checking for this remote ID the next time the MobiLink client requests synchronization for this remote ID. It may cause data inconsistency to reset a remote ID that did not have a successful synchronization in the last synchronization.

## 1.14.2.54 ml\_server\_delete System Procedure

This procedure is for internal use only.

## 1.14.2.55 ml\_server\_update System Procedure

This procedure is for internal use only.

## 1.14.3 MobiLink Utilities

A set of utility programs are included with MobiLink server. Each of the utilities can be accessed from one or more of SQL Central, Interactive SQL, or at a command prompt.

The following utilities are included with the MobiLink server:

- MobiLink Stop utility (mlstop)
- MobiLink User Authentication utility (mluser)
- MobiLink Replay utility (mlreplay)
- MobiLink Generated Replay API utility (mlgenreplayapi)
- MobiLink Arbiter Server utility for Microsoft Windows (mlarbiter)
- MobiLink Arbiter Server utility for UNIX and Linux (mlarbiter.sh)
- MobiLink Arbiter Stop utility (mlarbstop)

### In this section:

#### [MobiLink Stop Utility \(mlstop\) \[page 648\]](#)

Stops the MobiLink server on the local computer.

#### [MobiLink User Authentication Utility \(mluser\) \[page 649\]](#)

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.

#### [MobiLink Replay Utility \(mlreplay\) \[page 651\]](#)

The mlreplay utility is a tool used to replay MobiLink protocol information that is recorded by the MobiLink server.

#### [MobiLink Generated Replay API Utility \(mlgenreplayapi\) \[page 657\]](#)

The mlgenreplayapi tool reads a recorded protocol file and generates the **MobiLink Replay API** for the schema in that file.

#### [MobiLink Arbiter Server Utility for Windows \(mlarbiter\) \[page 658\]](#)

The mlarbiter command starts the MobiLink arbiter server.

#### [MobiLink Arbiter Server Utility for UNIX/Linux \(mlarbiter.sh\) \[page 659\]](#)

The mlarbiter.sh command starts and stops the MobiLink arbiter server.

#### [MobiLink Arbiter Stop Utility \(mlarbstop\) \[page 660\]](#)

The mlarbstop command is used to stop the MobiLink arbiter server.

## Related Information

[MobiLink Client Utilities](#)

[UltraLite Utilities](#)

[Database Administration Utilities](#)

### 1.14.3.1 MobiLink Stop Utility (mlstop)

Stops the MobiLink server on the local computer.

#### ☰ Syntax

```
mlstop [ options ] [ name ]
```

Option	Description
@data	Reads options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used.  To protect information in the configuration file, you can use the File Hiding utility (dbfhide) to encode the contents of the configuration file
-h	Hard shutdown. MobiLink stops all synchronizations and exits. Some remotes may report an error.
-q	Quiet mode. This suppresses the banner.
-t time	Soft shutdown, with a hard shutdown after the specified time. time is a number followed by D, H, M, or S (for days, hours, minutes and seconds). For example, -t 10m specifies that the server should be shut down in 10 minutes or when current synchronizations complete, whichever is sooner. D, H, M, and S are not case sensitive.
-w	Waits for the MobiLink server to shut down before returning from the command.
name	If the MobiLink server is started using the -zs option, it must be shut down by specifying the same server name.



## 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.

## Related Information

[Configuration Files](#)

[File Hiding Utility \(dbfhide\)](#)

[-zs mlsrv17 Option \[page 108\]](#)

## 1.14.3.2 MobiLink User Authentication Utility (mluser)

Registers MobiLink users at the consolidated database. For SQL Anywhere remotes, the users must have previously been created at the remote databases with the `CREATE SYNCHRONIZATION USER` statement.

### Syntax

```
mluser [ options ] -c "connection-string"  
{ -f file | -u user [ -p password ] }
```

Option	Description
@data	Reads options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used.  To protect information in the configuration file, you can use the File Hiding utility (dbfhide) to encode the contents of the configuration file
-c "keyword=value;..."	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.
-d	Deletes the user name(s) specified by <code>-f</code> or <code>-u</code> . This option cannot be used with <code>mluser -r</code> option.

Option	Description
<code>-f filename</code>	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 <code>-f</code> or <code>-u</code> .
<code>-fips</code>	When set, mluser fails if support for FIPS-certified encryption is not installed.
<code>-n user authentication policy name</code>	Registers a MobiLink user with LDAP user authentication.
<code>-O filename</code>	Logs output messages to the specified file.
<code>-ot filename</code>	Truncate the message log file and then append output messages to it. The default is to send output to the screen.
<code>-pc collation-id</code>	<p>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.</p> <p>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.</p>
<code>-p password</code>	Password to associate with the user. This option can only be used with <code>-u</code> .
<code>-r remote-id</code>	<p>Use this option with <code>-u username</code> and mluser resets the synchronization state for the given remote ID and user name. The <code>last_upload_time</code> and <code>last_download_time</code> columns in the <code>ml_subscription</code> table are reset to their default values for the given username and remote ID. The default values for the <code>progress</code>, <code>last_upload_time</code>, and <code>last_download_time</code> columns are 0, '1900/01/01 00:00:00', and '1900/01/01 00:00:00', respectively.</p> <p>This option cannot be used with the mluser <code>-d</code> option.</p> <div style="border-left: 2px solid orange; padding-left: 10px; margin-top: 10px;"> <p><b>⚠ Caution</b></p> <p>This option resets the synchronization state information for the given username and remote ID and this action cannot be undone. After the synchronization status is reset, the MobiLink server always accepts the first synchronization request from the client without checking the last synchronization status.</p> </div>
<code>-U username</code>	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> .
<code>-v</code>	Specifies verbose logging.

## Remarks

Given a user/password pair, the `mluser` utility first attempts to add the user. If the user has already been added to the consolidated database, it attempts to update the password for that user.

There are alternative ways to register user names in the consolidated database:

- Use SQL Central.
- Specify the `-zu+` command line option with `mlsrv17`. 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.

## Related Information

[Configuration Files](#)

[MobiLink Users in a Synchronization System](#)

[Transport Layer Security](#)

[File Hiding Utility \(dbfhide\)](#)

[Alternate Collations](#)

[-zu mlsrv17 Option \[page 109\]](#)

[CREATE SYNCHRONIZATION USER Statement \[MobiLink\]](#)

### 1.14.3.3 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]
```

Option	Description
<code>@data</code>	<p>Reads options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used.</p> <p>To protect information in the configuration file, you can use the File Hiding utility (dbfhide) to encode the contents of the configuration file</p>
<code>-ap</code>	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.
<code>-f time_scale_factor</code>	A multiplier evenly applied to recorded times.
<code>-ldt last_download_time</code>	Specify the last download time to send to the MobiLink server during the replay session. If the recorded protocol being replayed contains multiple synchronizations (this is possible if a persistent connection was recorded) only the first last download time is replaced; the rest will be replaced by the last download time the MobiLink server sends mlreplay during the replay session. Even if the -ldt option is not used, mlreplay replaces the last download time in all but the first synchronization with the last download time received from the MobiLink server during the replay session. A last download time can also be specified using the simulated client information file (when the -sci option is used) or by the IdentifySimulatedClient callback when a DLL is provided.
<code>-ls</code>	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. mlreplay still logs this information before exiting even when this option is not specified.
<code>-n number_of_simulated_clients</code>	<p>The number of simulated clients to run. The minimum is 1.</p> <p>This option can be used with the -sci option when the number of simulated clients specified by -n 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 <i>x</i> number of simulated clients, to replay a protocol with 1 to <i>x</i> simulated clients.</p>
<code>-o file</code>	Log command line options and output messages to the specified file.
<code>-os size</code>	Limit the maximum size of a message log file. When the log reaches the specified size (minimum 10 KB), it is renamed to <code>YYMMDDxx.rlg</code> and a new log file is started with the original name.

Option	Description
<code>-ot file</code>	Truncate the message log file. Log command line options and output messages to the specified file.
<code>-p password</code>	Replace passwords with the given password.
<code>-ping seconds</code>	<p>Ping a MobiLink server to determine whether the server is ready to receive synchronizations. By default, mlreplay pings the server for 60 seconds.</p> <p>If the <code>-ping</code> option is used, the following return codes are valid:</p> <p><b>-1</b></p> <p>Indicates that an error occurred.</p> <p><b>0</b></p> <p>Indicates that mlreplay was able to ping the server and the server is ready to receive synchronizations.</p> <p><b>1</b></p> <p>Indicates that mlreplay tried to ping the server but got no response; therefore, the server is not ready to receive synchronizations.</p>
<code>-r remote ID</code>	Replace remote IDs with the given remote ID. This option cannot be used with the <code>-rg</code> option.
<code>-rep number_of_repetitions</code>	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 <code>GetUploadTransaction</code> , <code>GetDownloadApplyTime</code> , <code>ReportEndOfReplay</code> , and <code>DelayStartOfReplay</code> callbacks are called for each repetition.
<code>-rg</code>	Replace remote IDs with a GUID.
<code>-rnt seconds</code>	<p>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.</p> <p>When specified, the <code>numRepetitions</code> parameter of any API callbacks is set to 0</p>
<code>-rp pattern</code>	Replace the given pattern in usernames, passwords, and remote IDs specified on the command line with the simulated client number.

Option	Description
<code>-sci file</code>	<p>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.</p> <p>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.</p>
<code>-sv script version</code>	Replace script versions with the given script version.
<code>-U user name</code>	Replace user names with the given user name.
<code>-X stream(opts)</code>	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.

## 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.

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 upload 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.

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.

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.



## Related Information

[Configuration Files](#)

[MobiLink Replay C++ Callbacks \[page 570\]](#)

[File Hiding Utility \(dbfhide\)](#)

[MobiLink Generated Replay API Utility \(mlgenreplayapi\) \[page 657\]](#)

[-rp mlsrv17 Option \[page 80\]](#)

[-rrp mlsrv17 Option \[page 81\]](#)

[-lsc mlsrv17 Option \[page 65\]](#)

### 1.14.3.4 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
```

Option	Description
@data	Reads options from the specified environment variable or configuration file. If both exist with the same name, the environment variable is used.  To protect information in the configuration file, you can use the File Hiding utility (dbfhide) to encode the contents of the configuration file
-d directory	The directory to output the generated files.
-o file	Logs command line options and output messages to the specified file.
-os size	Limits the maximum size of a message log file. When the log reaches the specified size (minimum 10 KB), it is renamed to <code>YYMMDDxx.rlg</code> and a new message log file is started with the original name.
-ot file	Truncates the message 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 `mlreplaycallbacks.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.

## Related Information

[Configuration Files](#)

[MobiLink Replay C++ Callbacks \[page 570\]](#)

[File Hiding Utility \(dbfhide\)](#)

[MobiLink Replay Utility \(mlreplay\) \[page 651\]](#)

### 1.14.3.5 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 to 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.

```
mlsrv17 -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.

## Related Information

[MobiLink Features and Architecture](#)

[-ca mlsrv17 Option \[page 54\]](#)

[-lsc mlsrv17 Option \[page 65\]](#)

### 1.14.3.6 MobiLink Arbiter Server Utility for UNIX/Linux (`mlarbiter.sh`)

The `mlarbiter.sh` command starts and stops the MobiLink arbiter server.

#### Syntax

```
mlarbiter.sh [ option ]
```

Option	Description
<code>start</code>	Starts the MobiLink Arbiter Server utility.
<code>stop</code>	Stops the MobiLink Arbiter Server utility.

## Remarks

The MobiLink arbiter listens on port 4953 by default.

This command is used with the `mlsrv17 -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 to 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.

```
mlsrv17 -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.

## Related Information

[MobiLink Features and Architecture](#)

[-ca mlsrv17 Option \[page 54\]](#)

[-lsc mlsrv17 Option \[page 65\]](#)

### 1.14.3.7 MobiLink Arbiter Stop Utility (mlarbstop)

The `mlarbstop` command is used to stop the MobiLink arbiter server.

☰ Syntax

```
mlarbstop [ option ]
```

Option	Description
-y	Stops the MobiLink arbiter server immediately, even if there are connections to it.

## 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
```

## Related Information

[MobiLink Features and Architecture](#)

[MobiLink Arbiter Server Utility for Windows \(mlarbiter\) \[page 658\]](#)

[MobiLink Arbiter Server Utility for UNIX/Linux \(mlarbiter.sh\) \[page 659\]](#)

[-ca mlsrv17 Option \[page 54\]](#)

[-lsc mlsrv17 Option \[page 65\]](#)

## 1.14.4 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.

Data type mappings between SQL Anywhere and/or UltraLite and the following supported consolidated databases are provided:

- Adaptive Server Enterprise
- IBM DB2 LUW (deprecated)
- Microsoft Azure
- Microsoft SQL Server
- MySQL
- Oracle
- SAP HANA
- SAP IQ Enterprise

### In this section:

#### [Adaptive Server Enterprise Data Mapping \[page 663\]](#)

SQL Anywhere and UltraLite remote data types can be mapped to Adaptive Server Enterprise consolidated data types and vice versa.

#### [IBM DB2 LUW Data Mapping \(Deprecated\) \[page 671\]](#)

SQL Anywhere and UltraLite remote data types can be mapped to IBM DB2 LUW consolidated data types and vice versa.

#### [Microsoft SQL Server Data Mapping \[page 679\]](#)

SQL Anywhere and UltraLite remote data types can be mapped to Microsoft SQL Server consolidated data types and vice versa.

#### [MySQL Data Mapping \[page 686\]](#)

SQL Anywhere and UltraLite remote data types can be mapped to MySQL consolidated data types and vice versa.

#### [Oracle Data Mapping \[page 691\]](#)

SQL Anywhere and UltraLite remote data types can be mapped to Oracle consolidated data types and vice versa.

#### [SAP HANA Data Mapping \[page 701\]](#)

SQL Anywhere and UltraLite remote data types can be mapped to SAP HANA consolidated data types and vice versa.

#### [SAP IQ Enterprise Data Mapping \[page 705\]](#)

SQL Anywhere and UltraLite remote data types can be mapped to SAP IQ Enterprise consolidated data types and vice versa.

## 1.14.4.1 Adaptive Server Enterprise Data Mapping

SQL Anywhere and UltraLite remote data types can be mapped to Adaptive Server Enterprise consolidated data types and vice versa.

### 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.

SQL Anywhere or UltraLite data type	Adaptive Server Enterprise data type	Notes
BIGINT	BIGINT	
BIT	BIT	
BINARY( $n \leq \text{MCL}$ )	BINARY( $n$ )	
BINARY( $n > \text{MCL}$ )	IMAGE	
CHAR( $n \leq \text{MCL}$ )	VARCHAR( $n$ )	
CHAR( $n > \text{MCL}$ )	TEXT	On download, ensure the values are not too long.
DATE	DATE	For Adaptive Server Enterprise DATE-TIME, the year must be in the range 1753-9999. For SQL Anywhere and UltraLite, the time value must in the format 00:00:00.

SQL Anywhere or UltraLite data type	Adaptive Server Enterprise data type	Notes
DATETIME	DATETIME <sup>1</sup> or BIGDATETIME <sup>2</sup>	<p>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.</p> <p>For download, SQL Anywhere keeps the original values from Adaptive Server Enterprise, but for upload, the values may not be exactly the original values.</p> <p>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.</p>
DECIMAL(p<39, s)	DECIMAL(p,s)	The precision of the Adaptive Server Enterprise NUMERIC can be from 1 to 38 digits (p<39).
DECIMAL(p>=39,s)		There is no corresponding data type in Adaptive Server Enterprise.
DOUBLE	DOUBLE PRECISION	
FLOAT(p)	FLOAT(p)	
IMAGE	IMAGE	
INTEGER	INTEGER	
LONG BINARY	IMAGE	
LONG NVARCHAR	UNITEXT	
LONG VARBIT	TEXT	
LONG VARCHAR	TEXT	
MONEY	MONEY	
NCHAR(c=<=MCL)	UNIVARCHAR(c/2)	
NCHAR(c>MCL)	UNITEXT	On download, ensure the values are not too long.



SQL Anywhere or UltraLite data type	Adaptive Server Enterprise data type	Notes
NTEXT	UNITEXT	
NUMERIC( <i>p</i> <39, <i>s</i> )	NUMERIC( <i>p</i> , <i>s</i> )	The precision of the Adaptive Server Enterprise decimal can be from 1 to 38 digits ( <i>p</i> <39).
NUMERIC( <i>p</i> >=39, <i>s</i> )		There is no corresponding data type in Adaptive Server Enterprise.
NVARCHAR( <i>c</i> <=MCL)	UNIVARCHAR( <i>c</i> /2)	
NVARCHAR( <i>c</i> >MCL)	UNIVARCHAR( <i>c</i> /2)	
REAL	REAL	
SMALLDATETIME	DATETIME <sup>1</sup> or BIGDATETIME <sup>2</sup>	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. However, SQL Anywhere or UltraLite SMALLDATETIME is accurate to the microsecond. Furthermore, the DATE part of a DATETIME in ASE can be any dates between January 1, 1753 and December 31, 9999, but the DATE part of a SMALLDATETIME in SQL Anywhere and UltraLite is restricted to dates between January 1, 1900 and June 6, 2079. Therefore any DATETIME values entered in an Adaptive Server Enterprise database or a SQL Anywhere/ UltraLite database must be restricted to satisfy these differences when these values are involved in synchronization. Otherwise, data inconsistency may occur.
SMALLINT	SMALLINT	
SMALLMONEY	SMALLMONEY	
TEXT	TEXT	

SQL Anywhere or UltraLite data type	Adaptive Server Enterprise data type	Notes
TIME	TIME <sup>1</sup> or BIGTIME <sup>2</sup>	<p>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.</p>
TIMESTAMP	DATETIME <sup>1</sup> or BIGDATETIME <sup>2</sup>	<p>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.</p> <p>For download, SQL Anywhere keeps the original values from Adaptive Server Enterprise, but for upload, the values may not be exactly the original values.</p> <p>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.</p>

SQL Anywhere or UltraLite data type	Adaptive Server Enterprise data type	Notes
TIMESTAMP WITH TIME ZONE	VARCHAR(34)	There is no equivalent data type in Adaptive Server Enterprise. Therefore, a TIMESTAMP WITH TIME ZONE column should be mapped to a VARCHAR(34) column. In upload, the MobiLink server first converts the data to a string using the format YYYY-MM-DD HH:NN:SS.SSSSSS [+-]HH:NN and then applies it to the consolidated database. In download, it converts the data from string to TIMESTAMP WITH TIME ZONE. Ensure the data in the consolidated database follows this format or the download will fail.
TINYINT	TINYINT	
UNIQUEIDENTIFIER	CHAR(36)	
UNIQUEIDENTIFIERSTR	CHAR(36)	Do not use UNIQUEIDENTIFIERSTR. Use UNIQUEIDENTIFIER instead.
UNSIGNED BIGINT	UNSIGNED BIGINT	
UNSIGNED INTEGER	UNSIGNED INT	
UNSIGNED SMALLINT	UNSIGNED SMALLINT	
UNSIGNED TINYINT	TINYINT	
VARBINARY( $n \leq MCL$ )	VARBINARY	
VARBINARY( $n > MCL$ )	IMAGE	
VARBIT( $n \leq MCL$ )	VARCHAR( $n$ )	
VARBIT( $n > MCL$ )	TEXT	
VARCHAR( $n \leq MCL$ )	VARCHAR( $n$ )	
VARCHAR( $n > MCL$ )	TEXT	
XML	TEXT	

<sup>1</sup> Only applies to Adaptive Server Enterprise before version 15.5.

<sup>2</sup> Only applies to Adaptive Server Enterprise version 15.5 and later.

## Mapping to SQL Anywhere or UltraLite Remote Data Types

The following table identifies how Adaptive Server Enterprise consolidated data types are mapped to SQL Anywhere and UltraLite remote data types. For example, a column of type DOUBLE PRECISION on the consolidated database should be type DOUBLE on the remote database.

Adaptive Server Enterprise data type	SQL Anywhere or UltraLite data type	Notes
BIGINT	BIGINT	
BIGDATETIME <sup>1</sup>	TIMESTAMP	
BIGTIME <sup>1</sup>	TIME	
BINARY( <i>n</i> )	BINARY( <i>n</i> )	
BIT	BIT	
CHAR( <i>n</i> )	VARCHAR( <i>n</i> )	There is no equivalence between SQL Anywhere CHAR/NCHAR and Adaptive Server Enterprise CHAR/NCHAR. SQL Anywhere CHAR/NCHAR is equivalent to VARCHAR/NVARCHAR. You should not use CHAR/NCHAR in a consolidated database column that is synchronized. If you must use non-SQL Anywhere CHAR/NCHAR, run the MobiLink server with the -b option.
DATE	DATE	For SQL Anywhere and UltraLite, the time value must in the format 00:00:00.
DATETIME	DATETIME	<p>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.</p> <p>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.</p>

Adaptive Server Enterprise data type	SQL Anywhere or UltraLite data type	Notes
DECIMAL(p,s)	DECIMAL(p,s)	
DOUBLE PRECISION	DOUBLE	
FLOAT(p)	FLOAT(p)	
IMAGE	LONG BINARY	
INT	INT	
MONEY	MONEY	
NCHAR(n)	VARCHAR(n)	The Adaptive Server Enterprise NCHAR and NVARCHAR store multibyte national character strings, they are different from SQL Anywhere NCHAR and NVARCHAR. In a multibyte environment, use SQL Anywhere or UltraLite VARCHAR.
NUMERIC(p,s)	NUMERIC(p,s)	
NVARCHAR(n)	VARCHAR(n)	The Adaptive Server Enterprise NCHAR and NVARCHAR store multibyte national character strings, they are different from SQL Anywhere NCHAR and NVARCHAR. In a multibyte environment, use SQL Anywhere or UltraLite VARCHAR.
REAL	REAL	
SMALLDATETIME	SMALLDATETIME	The Adaptive Server Enterprise SMALLDATETIME is accurate to the minute. 29.998 seconds or lower are rounded down to the nearest minute; values with 29.999 seconds or higher are rounded up to the nearest minute. SQL Anywhere or UltraLite SMALLDATETIME is accurate to the microsecond. To successfully synchronize, SQL Anywhere or UltraLite SMALLDATETIME must be rounded to the minute. Also, the year must be in the range 1900-2078.
SMALLINT	SMALLINT	
SMALLMONEY	SMALLMONEY	

Adaptive Server Enterprise data type	SQL Anywhere or UltraLite data type	Notes
TEXT	LONG VARCHAR	
TIME	TIME	<p>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.</p> <p>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, round the fractional second to 10 milliseconds.</p>
TIMESTAMP	VARBINARY(8)	<p>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.</p> <p>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.</p>
TINYINT	TINYINT	
UNSIGNED BIGINT	UNSIGNED BIGINT	
UNSIGNED INT	UNSIGNED INT	
UNSIGNED SMALLINT	UNSIGNED SMALLINT	
VARBINARY( <i>n</i> )	VARBINARY( <i>n</i> )	
VARCHAR( <i>n</i> )	VARCHAR( <i>n</i> )	
UNICHAR( <i>n</i> )	NVARCHAR( <i>n</i> )	Not available in UltraLite.

Adaptive Server Enterprise data type	SQL Anywhere or UltraLite data type	Notes
UNITEXT	LONG NVARCHAR	Not available in UltraLite.
UNIVARCHAR( <i>n</i> )	NVARCHAR( <i>n</i> )	Not available in UltraLite.

<sup>1</sup> Only applies to Adaptive Server Enterprise version 15.5 and later.

## 1.14.4.2 IBM DB2 LUW Data Mapping (Deprecated)

SQL Anywhere and UltraLite remote data types can be mapped to IBM DB2 LUW consolidated data types and vice versa.

### Mapping to IBM DB2 LUW Consolidated Data Types

The following table identifies how SQL Anywhere and UltraLite remote data types are mapped to IBM DB2 LUW consolidated data types. For example, a column of type BIT on the remote database should be type SMALLINT on the consolidated database.

When creating an IBM DB2 LUW table, you need to pay attention to the DB2 page size. IBM DB2 LUW has a maximum row length (MRL) based on the page size: the MRL is 4005 when the page size is 4K, 8101 when 8K, 16293 when 16K and 32677 when 32K. The length of all columns in a table cannot exceed the above limitation. If a table has a BLOB or CLOB column, you count row length using the LOB locator, not BLOB or CLOB data directly. For details, see the IBM DB2 LUW documentation.

SQL Anywhere or UltraLite data type	IBM DB2 LUW data type	Notes
BIGINT	BIGINT	
BINARY( <i>n</i> <MRL)	VARCHAR( <i>n</i> ) FOR BIT DATA	
BINARY( <i>n</i> >=MRL)	BLOB( <i>n</i> )	
BIT	SMALLINT	
CHAR( <i>n</i> <MRL)	VARCHAR( <i>n</i> )	
CHAR( <i>n</i> >=MRL)	CLOB( <i>n</i> )	IBM DB2 LUW values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
DATE	DATE	For SQL Anywhere and UltraLite, the time value must in the format 00:00:00.

SQL Anywhere or UltraLite data type	IBM DB2 LUW data type	Notes
DATETIME	TIMESTAMP	
DECIMAL(p<32,s)	DECIMAL(p,s)	The precision of SQL Anywhere DECIMAL is between 1 and 127. The maximum precision of IBM DB2 LUW DECIMAL is 31.
DECIMAL(p>=32,s)		Any data of SQL Anywhere DECIMAL precision greater than 31 cannot be synchronized to IBM DB2 LUW.
DOUBLE	DOUBLE	DOUBLE is an imprecise numeric data type that is subject to rounding. When working with different types of computers, the underlying storage of DOUBLE is often different, resulting in different rounding. DOUBLE is a bad choice to use in a primary key because primary keys are looking for equality. This is especially true in a synchronization environment because the consolidated database often runs on different hardware from the remote database.
FLOAT(1-24)	REAL	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.
FLOAT(25-53)	DOUBLE	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.
IMAGE	BLOB(n)	
INTEGER	INTEGER	
LONG BINARY	BLOB(n)	



SQL Anywhere or UltraLite data type	IBM DB2 LUW data type	Notes
LONG NVARCHAR	CLOB(n)	There is no corresponding data type in IBM DB2 LUW. If the IBM DB2 LUW character set is Unicode, SQL Anywhere LONG NVARCHAR can synchronize to IBM DB2 CLOB. UltraLite doesn't have LONG NVARCHAR.
LONG VARBIT	CLOB(n)	
LONG VARCHAR	CLOB(n)	
MONEY	DECIMAL(19,4)	
NCHAR(c)	VARCHAR(n) or CLOB(n)	There is no corresponding data type in IBM DB2 LUW. If the IBM DB2 LUW character set is Unicode, NCHAR can synchronize to IBM DB2 LUW VARCHAR or CLOB. The size of SQL Anywhere NCHAR is characters and the size of IBM DB2 LUW VARCHAR is bytes. If you map to VARCHAR, the total bytes of NCHAR 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).
NUMERIC(p<32,s)	NUMERIC(p,s)	
NUMERIC(p>=32,s)		There is no corresponding data type in IBM DB2 LUW.
NTEXT	CLOB(n)	There is no corresponding data type in IBM DB2 LUW. If the IBM DB2 LUW character set is Unicode, NTEXT can synchronize to IBM DB2 LUW CLOB.

SQL Anywhere or UltraLite data type	IBM DB2 LUW data type	Notes
NVARCHAR( <i>c</i> )	VARCHAR( <i>n</i> ) or CLOB( <i>n</i> )	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( <i>c</i> ), but it is approximately $c = n/4$ . In general, if <i>c</i> is less than MRL/4, map to VARCHAR( <i>n</i> ), but if <i>c</i> is greater than or equal to MRL/4, map to CLOB( <i>n</i> ).
REAL	REAL	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.
SMALLDATETIME	TIMESTAMP	
SMALLINT	SMALLINT	
SMALLMONEY	DECIMAL(10,4)	
ST_GEOMETRY	ST_GEOMETRY	
TEXT	CLOB( <i>n</i> )	

SQL Anywhere or UltraLite data type	IBM DB2 LUW data type	Notes
TIME	TIMESTAMP or TIME	SQL Anywhere and UltraLite TIME values with fractional seconds require IBM DB2 LUW TIMESTAMP. SQL Anywhere and UltraLite time values with fractional seconds that are always zero can use IBM DB2 TIME. To preserve the precision of a time column, the MobiLink server always binds the TIME column with the ODBC SQL_TYPE_TIMESTAMP data type. When the consolidated database is running on a DB2 9.7 server, you may need to use DB2 conversion functions to explicitly convert the column between TIMESTAMP and TIME if the column is a part of a primary key.
TIMESTAMP	TIMESTAMP	
TIMESTAMP WITH TIME ZONE	VARCHAR(34)	There is no equivalent data type in IBM DB2 LUW. Therefore, a TIMESTAMP WITH TIME ZONE column should be mapped to a VARCHAR(34) column. In upload, the MobiLink server first converts the data to a string using the format YYYY-MM-DD HH:NN:SS.SSSSSS [+-]JHH:NN and then applies it to the consolidated database. In download, it converts the data from string to TIMESTAMP WITH TIME ZONE. Ensure the data in the consolidated database follows this format or the download will fail.
TINYINT	SMALLINT	For download, IBM DB2 LUW values must be non-negative.
UNIQUEIDENTIFIER	CHAR(36)	
UNIQUEIDENTIFIERSTR	CHAR(36)	UNIQUEIDENTIFIERSTR is not recommended for IBM DB2 LUW. Use UNIQUEIDENTIFIER instead.
UNSIGNED BIGINT	DECIMAL(20)	For download, IBM DB2 LUW values must be non-negative.
UNSIGNED INTEGER	DECIMAL(11)	For download, IBM DB2 LUW values must be non-negative.
UNSIGNED SMALLINT	DECIMAL(5)	For download, IBM DB2 LUW values must be non-negative.

SQL Anywhere or UltraLite data type	IBM DB2 LUW data type	Notes
UNSIGNED TINYINT	SMALLINT	For download, IBM DB2 LUW values must be non-negative.
VARBINARY( $n < \text{MRL}$ )	VARCHAR( $n$ ) FOR BIT DATA	
VARBINARY( $n \geq \text{MRL}$ )	BLOB( $n$ )	
VARBIT( $n < \text{MRL}$ )	VARCHAR( $n$ )	
VARBIT( $n \geq \text{MRL}$ )	CLOB( $n$ )	
VARCHAR( $n < \text{MRL}$ )	VARCHAR( $n$ )	
VARCHAR( $n \geq \text{MRL}$ )	CLOB( $n$ )	IBM DB2 LUW values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
XML	CLOB( $n$ )	

## Mapping to SQL Anywhere or UltraLite Remote Data Types

The following table identifies how IBM DB2 LUW consolidated data types are mapped to SQL Anywhere and UltraLite remote data types. For example, a column of type INT on the consolidated database should be type INTEGER on the remote database.

When creating an IBM DB2 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.

IBM DB2 LUW data type	SQL Anywhere or UltraLite data type	Notes
BLOB	LONG BINARY	
BIGINT	BIGINT	
CHAR( $n$ )	VARCHAR( $n$ )	There is no equivalent to IBM DB2 LUW CHAR in SQL Anywhere. You should not use CHAR in a consolidated database column that is synchronized. If you must synchronize IBM DB2 LUW CHAR columns, run MobiLink server with the -b option.
CHAR( $n$ ) FOR BIT DATA	BINARY( $n$ )	

IBM DB2 LUW data type	SQL Anywhere or UltraLite data type	Notes
CLOB( <i>n</i> )	LONG VARCHAR	
DATE	DATE	For SQL Anywhere and UltraLite, the time value must in the format 00:00:00.
DB2GSE.ST_GEOMETRY	ST_GEOMETRY	
DBCLOB( <i>n</i> )	LONG VARCHAR	The data type DBCLOB( <i>n</i> ) 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( <i>n</i> ) is equivalent to CLOB.
DECIMAL( <i>p,s</i> )	DECIMAL( <i>p,s</i> )	
DOUBLE	DOUBLE	DOUBLE is an imprecise numeric data type that is subject to rounding. When working with different types of computers, the underlying storage of DOUBLE is often different, resulting in different rounding. DOUBLE is a bad choice to use in a primary key because primary keys are looking for equality. This is especially true in a synchronization environment because the consolidated database often runs on different hardware from the remote database.
FLOAT	DOUBLE	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.
GRAPHIC( <i>n</i> )	VARCHAR(2 <i>n</i> )	IBM DB2 LUW GRAPHIC does blank-padding, but SQL Anywhere CHAR does not. 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.

IBM DB2 LUW data type	SQL Anywhere or UltraLite data type	Notes
INT	INTEGER	
LONG VARCHAR	VARCHAR(32700)	
LONG VARCHAR FOR BIT DATA	VARBINARY(32700)	
LONG VARGRAPHIC( <i>n</i> )	VARCHAR(32700)	The data type LONG VARGRAPHIC is only used for double-byte characters. SQL Anywhere does not have a corresponding data type. When the IBM DB2 LUW character set is Unicode, LONG VARGRAPHIC is equivalent LONG VARCHAR.
NUMERIC( <i>p,s</i> )	NUMERIC( <i>p,s</i> )	
REAL	REAL	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.
SMALLINT	SMALLINT	
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_TIME-STAMP data type. When the consolidated database is running on a DB2 9.7 server, you may need to use DB2 conversion functions to explicitly convert the column between TIMESTAMP and TIME if the column is a part of a primary key.
TIMESTAMP	TIMESTAMP	
VARCHAR( <i>n</i> )	VARCHAR( <i>n</i> )	
VARCHAR( <i>n</i> ) FOR BIT DATA	VARBINARY( <i>n</i> )	

IBM DB2 LUW data type	SQL Anywhere or UltraLite data type	Notes
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.

### 1.14.4.3 Microsoft SQL Server Data Mapping

SQL Anywhere and UltraLite remote data types can be mapped to Microsoft SQL Server consolidated data types and vice versa.

#### 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.

SQL Anywhere or UltraLite data type	Microsoft SQL Server data type	Notes
BIGINT	BIGINT	
BINARY(n<=8000)	VARBINARY(n)	
BINARY(n>8000)	VARBINARY(MAX)	
BIT	BIT	
CHAR(n<=8000)	VARCHAR(n)	
CHAR(n>8000)	VARCHAR(MAX)	
DATE	DATE	
DATETIME	DATETIME2	Microsoft SQL Server DATETIME2 and TIME values are accurate to 100 nanoseconds. However, TIMESTAMP and TIME values are only accurate to 1 microsecond. To successfully synchronize DATETIME2 and TIME, round the fractional second to 1 microsecond.

SQL Anywhere or UltraLite data type	Microsoft SQL Server data type	Notes
DECIMAL( <i>p</i> ≤38, <i>s</i> )	DECIMAL( <i>p</i> , <i>s</i> )	Microsoft SQL Server DECIMAL/ NUMERIC precision ranges from 1 to 38, so <i>p</i> must be less than 39.
DECIMAL( <i>p</i> >38, <i>s</i> )		There is no corresponding data type in Microsoft SQL Server.
DOUBLE	FLOAT(53)	
FLOAT( <i>p</i> )	FLOAT( <i>p</i> )	
IMAGE	VARBINARY(MAX)	
INTEGER	INT	
LONG BINARY	VARBINARY(MAX)	
LONG NVARCHAR	NVARCHAR(MAX)	
LONG VARBIT	VARCHAR(MAX)	
LONG VARCHAR	VARCHAR(MAX)	
MONEY	MONEY	
NCHAR( <i>n</i> ≤4000)	NVARCHAR( <i>c</i> )	
NCHAR( <i>n</i> >4000)	NVARCHAR(MAX)	
NTEXT	NVARCHAR(MAX)	
NUMERIC( <i>p</i> ≤38, <i>s</i> )	NUMERIC( <i>p</i> , <i>s</i> )	Microsoft SQL Server DECIMAL/ NUMERIC precision ranges from 1 to 38, so <i>p</i> must be less than 39.
NUMERIC( <i>p</i> >38, <i>s</i> )		There is no corresponding data type in Microsoft SQL Server.
NVARCHAR( <i>n</i> ≤4000)	NVARCHAR( <i>c</i> )	
NVARCHAR( <i>n</i> >4000)	NVARCHAR(MAX)	
REAL	REAL	



SQL Anywhere or UltraLite data type	Microsoft SQL Server data type	Notes
SMALLDATETIME	SMALLDATETIME	SQL Anywhere and UltraLite SMALLDATETIME is implemented as TIME-STAMP. Microsoft SQL Server SMALLDATETIME is accurate to the minute. 29.998 seconds or lower are rounded down to the nearest minute; values with 29.999 seconds or higher are rounded up to the nearest minute. SQL Anywhere or UltraLite SMALLDATETIME is accurate to the microsecond. To successfully synchronize, SQL Anywhere or UltraLite SMALLDATETIME must be rounded to the minute. The year must be in the range 1900-2078.
SMALLINT	SMALLINT	
SMALLMONEY	SMALLMONEY	
ST_GEOMETRY	GEOMETRY	
TEXT	VARCHAR(MAX)	
TIME	TIME	Microsoft SQL Server DATETIME2 and TIME values are accurate to 100 nanoseconds. However, TIMESTAMP and TIME values are only accurate to 1 microsecond. To successfully synchronize DATETIME2 and TIME, round the fractional second to 1 microsecond.
TIMESTAMP	DATETIME2	Microsoft SQL Server DATETIME2 and TIME values are accurate to 100 nanoseconds. However, TIMESTAMP and TIME values are only accurate to 1 microsecond. To successfully synchronize DATETIME2 and TIME, round the fractional second to 1 microsecond.
TIMESTAMP WITH TIME ZONE	DATETIMEOFFSET	
TINYINT	TINYINT	For download, values must be non-negative.
UNIQUEIDENTIFIER	UNIQUEIDENTIFIER	
UNIQUEIDENTIFIERSTR	UNIQUEIDENTIFIER	

SQL Anywhere or UltraLite data type	Microsoft SQL Server data type	Notes
UNSIGNED BIGINT	NUMERIC(20)	For download, values must be non-negative.
UNSIGNED INTEGER	NUMERIC(11)	For download, values must be non-negative.
UNSIGNED TINYINT	TINYINT	For download, values must be non-negative.
UNSIGNED SMALLINT	INT	For download, values must be non-negative.
VARBINARY( <i>n</i> ≤8000)	VARBINARY( <i>n</i> )	
VARBINARY( <i>n</i> >8000)	VARBINARY(MAX)	
VARBIT( <i>n</i> ≤8000)	VARCHAR( <i>n</i> )	
VARBIT( <i>n</i> >8000)	VARCHAR(MAX)	
VARCHAR( <i>n</i> ≤8000)	VARCHAR( <i>c</i> )	
VARCHAR( <i>n</i> >8000)	VARCHAR(MAX)	
XML	XML or VARCHAR(MAX)	For Microsoft SQL Server 2005, use XML. For other versions, use VARCHAR(MAX).

## Mapping to SQL Anywhere or UltraLite Remote Data Types

The following table identifies how Microsoft SQL Server consolidated data types are mapped to SQL Anywhere and UltraLite remote data types. For example, a column of type TEXT on the remote database should be type LONG VARCHAR on the consolidated database.

Microsoft SQL Server data type	SQL Anywhere or UltraLite data type	Notes
BIGINT	BIGINT	
BINARY( <i>n</i> )	BINARY( <i>n</i> )	
BIT	BIT	

Microsoft SQL Server data type	SQL Anywhere or UltraLite data type	Notes
CHAR( <i>n</i> )	VARCHAR( <i>n</i> )	A Microsoft SQL Server CHAR column is blank padded. A SQL Anywhere CHAR column is not blank padded by default and is equivalent to a VARCHAR column. Therefore, try to avoid using the CHAR data type in the synchronization tables in Microsoft SQL Server. If you must use the CHAR data type in the Microsoft SQL Server consolidated database, run the MobiLink server with the -b command line option to help resolve the differences between SQL Anywhere CHAR and non-SQL Anywhere CHAR.
DATE	DATE	
DATETIME	TIMESTAMP or DATETIME	Microsoft SQL Server DATETIME values are accurate to 1/300 second. The last digit of the fractional second is always 0, 3, or 6. Other digits are rounded to one of these three digits, so, 0 and 1 round to 0; 2, 3, and 4 round to 3; 5, 6, 7, and 8 round to 6; and 9 rounds to 10. For download, SQL Anywhere keeps the original values from Microsoft SQL Server, but for upload, the values may not be exactly the original values. If DATETIME is used for a primary key, conflict resolution may fail. To successfully synchronize DATETIME, you should round the fractional second to 10 milliseconds. The year must be in the range 1753-9999.
DATETIME2	TIMESTAMP	Microsoft SQL Server DATETIME2 and TIME values are accurate to 100 nanoseconds. However, TIMESTAMP and TIME values are only accurate to 1 microsecond. To successfully synchronize DATETIME2 and TIME, it is recommended that you round the fractional second to 1 microsecond.
DECIMAL( <i>p,s</i> )	DECIMAL( <i>p,s</i> )	
FLOAT( <i>p</i> )	FLOAT( <i>p</i> )	

Microsoft SQL Server data type	SQL Anywhere or UltraLite data type	Notes
GEOMETRY	ST_GEOMETRY	
IMAGE	LONG BINARY	
INT	INT	
MONEY	MONEY	
NCHAR( <i>n</i> )	NVARCHAR( <i>c</i> )	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.
NTEXT	LONG NVARCHAR	Not available in UltraLite.
NVARCHAR( <i>c</i> )	NVARCHAR( <i>c</i> )	Not available in UltraLite.
NVARCHAR(MAX)	LONG NVARCHAR	Not available in UltraLite.
NUMERIC( <i>p,s</i> )	NUMERIC( <i>p,s</i> )	
REAL	REAL	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.

Microsoft SQL Server data type	SQL Anywhere or UltraLite data type	Notes
SMALLDATETIME	SMALLDATETIME	SQL Anywhere and UltraLite SMALLDATETIME is implemented as TIME-STAMP. Microsoft SQL Server SMALLDATETIME is accurate to the minute. 29.998 seconds or lower are rounded down to the nearest minute; values with 29.999 seconds or higher are rounded up to the nearest minute. SQL Anywhere or UltraLite SMALLDATETIME is accurate to the microsecond. To successfully synchronize, SQL Anywhere or UltraLite SMALLDATETIME must be rounded to the minute. The year must be in the range 1900-2078.
SMALLINT	SMALLINT	
SMALLMONEY	SMALLMONEY	
TEXT	LONG VARCHAR	
TIME	TIME	Microsoft SQL Server DATETIME2 and TIME values are accurate to 100 nanoseconds. However, TIMESTAMP and TIME values are only accurate to 1 microsecond. To successfully synchronize DATETIME2 and TIME, round the fractional second to 1 microsecond.
TIMESTAMP	VARBINARY(8)	<p>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.</p> <p>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.</p>
DATETIMEOFFSET	TIMESTAMP WITH TIME ZONE	
TINYINT	TINYINT	

Microsoft SQL Server data type	SQL Anywhere or UltraLite data type	Notes
UNIQUEIDENTIFIER	UNIQUEIDENTIFIER	
VARBINARY( <i>n</i> )	VARBINARY( <i>n</i> )	
VARBINARY(MAX)	LONG BINARY	
VARCHAR( <i>n</i> )	VARCHAR( <i>n</i> )	
VARCHAR(MAX)	LONG VARCHAR	
XML	XML	

## 1.14.4.4 MySQL Data Mapping

SQL Anywhere and UltraLite remote data types can be mapped to MySQL consolidated data types and vice versa.

### Mapping to MySQL Consolidated Data Types

The following table identifies how SQL Anywhere and UltraLite remote data types are mapped to MySQL consolidated data types. For example, a column of type TEXT on the remote database should be type LONGTEXT on the consolidated database.

SQL Anywhere or UltraLite data type	MySQL data type	Notes
BIGINT	BIGINT	
BINARY( <i>n</i> <=255)	BINARY( <i>n</i> )	
BINARY( <i>n</i> >255)	BLOB	
BIT	BIT	
CHAR( <i>n</i> <=255)	CHAR( <i>n</i> )	
CHAR( <i>n</i> >255)	TEXT( <i>n</i> )	
DATE	DATE	The year must range from 1000 to 9999.
DATETIME	DATETIME	The MySQL DATETIME data type does not support fractional seconds. The year must range from 1000 to 9999.

SQL Anywhere or UltraLite data type	MySQL data type	Notes
DECIMAL(p<=65,s<=30)	DECIMAL(p,s)	
DECIMAL(p>65,s>30)		There is no corresponding data type in MySQL if the precision is greater than 65 or if the scale is greater than 30.
DOUBLE	DOUBLE	
FLOAT	FLOAT	
IMAGE	LONGBLOB	
INTEGER	INTEGER	
LONG BINARY	LONGBLOB	
LONG NVARCHAR	LONGTEXT CHARACTER SET UTF8	
LONG VARBIT	LONGTEXT	
LONG VARCHAR	LONGTEXT	
MONEY	NUMERIC(19,4)	
NCHAR(n<=255)	CHAR(n) CHARACTER SET UTF8	
NCHAR(n>255)	TEXT CHARACTER SET UTF8	
NTEXT	LONGTEXT CHARACTER SET UTF8	
NUMERIC(p<=65,s<=30)	DECIMAL(p,s)	
NUMERIC(p>65,s>30)		There is no corresponding data type in MySQL.
NVARCHAR(n)	VARCHAR(n) CHARACTER SET UTF8	
REAL	REAL	
SMALLDATETIME	DATETIME	The MySQL DATETIME data type does not support fractional seconds. The year must range from 1000 to 9999.
SMALLINT	SMALLINT	
SMALLMONEY	NUMERIC(10,4)	
ST_GEOMETRY	GEOMETRY	
TEXT	LONGTEXT	

SQL Anywhere or UltraLite data type	MySQL data type	Notes
TIME	TIME	The MySQL TIME data type does not support fractional seconds.
TIMESTAMP	DATETIME	The MySQL DATETIME data type does not support fractional seconds. The year must range from 1000 to 9999.
TIMESTAMP WITH TIME ZONE	VARCHAR(34)	There is no equivalent data type in MySQL. Therefore, a TIMESTAMP WITH TIME ZONE column should be mapped to a VARCHAR(34) column. In upload, the MobiLink server first converts the data to a string using the format YYYY-MM-DD HH:NN:SS.SSSSSS [+-]HH:NN and then applies it to the consolidated database. In download, it converts the data from string to TIMESTAMP WITH TIME ZONE. Ensure the data in the consolidated database follows this format or the download will fail.
TINYINT	TINYINT UNSIGNED	TINYINT is always unsigned in SQL Anywhere and UltraLite.
UNIQUEIDENTIFIER	CHAR(36)	
UNIQUEIDENTIFIERSTR	CHAR(36)	
VARBINARY( <i>n</i> )	VARCHAR( <i>n</i> )	
VARBIT( <i>n</i> ≤8000)	VARCHAR( <i>n</i> )	
VARBIT( <i>n</i> >8000)	TEXT	
VARCHAR( <i>n</i> )	VARCHAR( <i>n</i> )	
XML	LONGTEXT	

## Mapping to SQL Anywhere or UltraLite Remote Data Types

The following table identifies how MySQL consolidated data types are mapped to SQL Anywhere and UltraLite remote data types. For example, a column of type BOOL on the consolidated database should be type BIT on the remote database.

MySQL data type	SQL Anywhere or UltraLite data type	Notes
BIGINT	BIGINT	



MySQL data type	SQL Anywhere or UltraLite data type	Notes
BINARY( <i>n</i> )	BINARY( <i>n</i> )	
BIT(1)	BIT	
BIT( <i>n</i> >1)	UNSIGNED BIGINT	
BLOB( <i>n</i> ≤32767)	VARBINARY( <i>n</i> )	
BLOB( <i>n</i> >32767)	IMAGE	
BOOL	BIT	
CHAR( <i>n</i> )	CHAR( <i>n</i> )	
DATE	DATE	The year must range from 1000 to 9999.
DATETIME	DATETIME	The MySQL DATETIME data type does not support fractional seconds. The year must range from 1000 to 9999.
DOUBLE	DOUBLE	
DECIMAL	DECIMAL	
ENUM		There is no corresponding data type in SQL Anywhere or UltraLite.
GEOMETRY	ST_GEOMETRY	
INTEGER	INTEGER	
LINestring		There is no corresponding data type in SQL Anywhere or UltraLite.
LOBLOB	IMAGE	
LONGTEXT	TEXT	
MEDIUMBLOB	IMAGE	
MEDIUMINT	INTEGER	
MEDIUMTEXT	TEXT	
MULTILINestring		There is no corresponding data type in SQL Anywhere or UltraLite.
MULTIPOINT		There is no corresponding data type in SQL Anywhere or UltraLite.

MySQL data type	SQL Anywhere or UltraLite data type	Notes
MULTIPOLYGON		There is no corresponding data type in SQL Anywhere or UltraLite.
NCHAR	NCHAR	Not available in UltraLite.
NUMERIC	NUMERIC	
NVARCHAR	NVARCHAR	Not available in UltraLite.
POINT		There is no corresponding data type in SQL Anywhere or UltraLite.
POLYGON		There is no corresponding data type in SQL Anywhere or UltraLite.
REAL	REAL	
SET		There is no corresponding data type in SQL Anywhere or UltraLite.
SMALLINT	SMALLINT	
TEXT( $n \leq 32767$ )	VARCHAR( $n$ )	
TEXT( $n > 32767$ )	TEXT	
TIME	TIME	The MySQL TIME data type does not support fractional seconds. The range of TIME in MySQL is '-838:59:59' to '838:59:59'. The range of TIME in SQL Anywhere or UltraLite is '00:00:00.000000' to '23:59:59.999999'.
TIMESTAMP	TIMESTAMP	The MySQL DATETIME data type does not support fractional seconds. The year must range from 1000 to 9999. Although MySQL offers automatic initialization and updating on TIMESTAMP columns, SQL Anywhere and UltraLite only offers automatic initialization.
TINYBLOB	VARBINARY	
TINYINT	SMALLINT	TINYINT is always unsigned in SQL Anywhere and UltraLite. Must be a positive value.
TINYINT UNSIGNED	TINYINT	TINYINT is always unsigned in SQL Anywhere and UltraLite.

MySQL data type	SQL Anywhere or UltraLite data type	Notes
TINYTEXT	VARCHAR	
VARBINARY( $n \leq 32767$ )	VARBINARY( $n$ )	
VARBINARY( $n > 32767$ )	IMAGE	
VARCHAR( $n \leq 32767$ )	VARCHAR( $n$ )	
VARCHAR( $n > 32767$ )	TEXT	
YEAR[(2 4)]	INTEGER	SQL Anywhere and UltraLite do not support the YEAR data type. YEAR needs to be mapped to INTEGER in a remote database. The INTEGER value must range from 1000 to 9999.

## 1.14.4.5 Oracle Data Mapping

SQL Anywhere and UltraLite remote data types can be mapped to Oracle consolidated data types and vice versa.

### Mapping to Oracle Consolidated Data Types

The following table identifies how SQL Anywhere and UltraLite remote data types are mapped to Oracle consolidated data types. For example, a column of type BIT on the remote database should be type NUMBER on the consolidated database.

SQL Anywhere or UltraLite data type	Oracle data type	Notes
BIGINT	NUMBER(20)	
BINARY( $n$ )	RAW( $n$ )	Only valid for Oracle 12.1 or later.
BINARY( $n \leq 2000$ )	RAW( $n$ )	
BINARY( $n > 2000$ )	BLOB	Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
BIT	NUMBER(1)	
CHAR( $n$ )	VARCHAR2( $n$ )	Only valid for Oracle 12.1 or later.

SQL Anywhere or UltraLite data type	Oracle data type	Notes
CHAR( $n \leq 4000$ )	VARCHAR2( $n$ byte)	Oracle VARCHAR2 allows you to specify the maximum number of bytes or characters. The maximum length of VARCHAR2 data is 4000 bytes. If you specify the character number, make sure the maximum data length is not over 4000 bytes.
CHAR( $n > 4000$ )	CLOB	Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
DATE	DATE	
DATETIME	TIMESTAMP	The year must be in the range 1-9999.
DECIMAL( $p \leq 38, s$ )	NUMBER( $p, 0 \leq s \leq 38$ )	In SQL Anywhere DECIMAL, $p$ is between 1 and 127, and $s$ is always less than or equal to $p$ . In Oracle NUMBER, $p$ ranges from 1 to 38, and $s$ ranges from -84 to 127. To synchronize, the Oracle NUMBER scale must be restricted to between 0 and 38.
DECIMAL( $p > 38, s$ )		There is no corresponding data type in Oracle.
DOUBLE	DOUBLE PRECISION or BINARY_DOUBLE <sup>1</sup>	The special values INF, -INF and NAN of Oracle Database 11g BINARY_FLOAT and BINARY_DOUBLE cannot be synchronized with SQL Anywhere or UltraLite.
FLOAT( $p$ )	FLOAT( $p$ )	
IMAGE	BLOB	Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
INTEGER	INT	
LONG BINARY	BLOB	Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.

SQL Anywhere or UltraLite data type	Oracle data type	Notes
LONG NVARCHAR	NCLOB	<p>Oracle CLOB and NCLOB can hold up to 4G of data. SQL Anywhere LONG VARCHAR and LONG NVARCHAR can only hold up to 2G.</p> <p>Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.</p>
LONG VARBIT	CLOB	<p>Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.</p>
LONG VARCHAR	CLOB	<p>Oracle CLOB and NCLOB can hold up to 4G of data. SQL Anywhere LONG VARCHAR and LONG NVARCHAR can only hold up to 2G.</p> <p>Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.</p>
MONEY	NUMBER(19,4)	
NCHAR(c)	NVARCHAR2(c) or NCLOB	<p>For versions of Oracle prior to 12.1, 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.</p> <p>For Oracle 12.1 or later, the size of SQL Anywhere NCHAR and NVARCHAR indicates the maximum number of Unicode characters, whereas the size of Oracle NVARCHAR2 indicates the maximum number of bytes. It is difficult to calculate the maximum byte length from character size. In general, if the size is over 16383, map to NCLOB, otherwise map to NVARCHAR2.</p>

SQL Anywhere or UltraLite data type	Oracle data type	Notes
NTEXT	NCLOB	<p>Oracle NCLOB can hold up to 4G of data. SQL Anywhere NTEXT (or LONG NVARCHAR) can only hold up to 2G.</p> <p>Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.</p>
NUMERIC( <i>p</i> ≤38, <i>s</i> )	NUMBER( <i>p</i> , 0≤ <i>s</i> ≤38)	<p>In SQL Anywhere NUMERIC, <i>p</i> is between 1 and 127, and <i>s</i> is always less than or equal to <i>p</i>. In Oracle NUMBER, <i>p</i> ranges from 1 to 38, and <i>s</i> ranges from -84 to 127. To synchronize, the Oracle NUMBER scale must be restricted to between 0 and 38.</p>
NUMERIC( <i>p</i> >38, <i>s</i> )		<p>There is no corresponding data type in Oracle.</p>

SQL Anywhere or UltraLite data type	Oracle data type	Notes
NVARCHAR	NVARCHAR2(c) or NCLOB	<p>The size of NVARCHAR2 is the maximum number of Unicode characters in Oracle and the size of NCHAR and NVARCHAR is the maximum number of characters in SQL Anywhere. A multi-byte character can take as many as four bytes to be stored in a SQL Anywhere database. In general, it is difficult to calculate maximum byte length from character size. For versions of Oracle prior to 12.1, NVARCHAR2 cannot be over 2000 unicode characters. Therefore, SQL Anywhere NVARCHAR(n) should be mapped to Oracle NVARCHAR2, when n&lt;= 1000. Otherwise, it should be mapped to NCLOB.</p> <p>For versions of Oracle prior to 12.1, 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.</p> <p>For Oracle 12.1 or later, the size of SQL Anywhere NCHAR and NVARCHAR indicates the maximum number of Unicode characters, whereas the size of Oracle NVARCHAR2 indicates the maximum number of bytes. It is difficult to calculate the maximum byte length from character size. In general, if the size is over 16383, map to NCLOB, otherwise map to NVARCHAR2.</p>
REAL	REAL or BINARY_FLOAT <sup>1</sup>	The special values INF, -INF and NAN of Oracle Database 11g BINARY_FLOAT and BINARY_DOUBLE cannot be synchronized with SQL Anywhere or UltraLite.
SMALLDATETIME	TIMESTAMP	The year must be in the range 1-9999.
SMALLINT	NUMBER(5)	

SQL Anywhere or UltraLite data type	Oracle data type	Notes
SMALLMONEY	NUMBER(10,4)	
ST_GEOMETRY	SDO_GEOMETRY	
TEXT	CLOB	Oracle CLOB can hold up to 4G of data. SQL Anywhere TEXT (or LONG VARCHAR) can only hold up to 2G.  Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
TIME	TIMESTAMP	
TIMESTAMP	TIMESTAMP	The year must be in the range 1-9999.
TIMESTAMP WITH TIME ZONE	TIMESTAMP WITH TIME ZONE	
TINYINT	NUMBER(3)	For download, Oracle values must be non-negative.
UNSIGNED BIGINT	NUMBER(20)	For download, Oracle values must be non-negative.
UNSIGNED INTEGER	NUMBER(11)	For download, Oracle values must be non-negative.
UNSIGNED SMALLINT	NUMBER(5)	For download, Oracle values must be non-negative.
UNSIGNED TINYINT	NUMBER(3)	For download, Oracle values must be non-negative.
UNIQUEIDENTIFIER	CHAR(36)	
UNIQUEIDENTIFIERSTR	CHAR(36)	UNIQUEIDENTIFIERSTR is not recommended to use for Oracle. Use UNIQUEIDENTIFIER instead.
VARBINARY( $n \leq 2000$ )	RAW( $n$ )	
VARBINARY( $n > 2000$ )	BLOB	Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
VARBIT( $n$ )	VARCHAR2( $n$ )	Only valid for Oracle 12.1 or later.
VARBIT( $n \leq 4000$ )	VARCHAR2( $n$ byte)	



SQL Anywhere or UltraLite data type	Oracle data type	Notes
VARBIT( $n > 4000$ )	CLOB	Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
VARCHAR( $n$ )	VARCHAR2( $n$ )	Only valid for Oracle 12.1 or later.
VARCHAR( $n \leq 4000$ )	VARCHAR2( $n$ byte)	Oracle VARCHAR2 allows you to specify the maximum number of bytes or characters. The maximum length of VARCHAR2 data is 4000 bytes. If you specify the character number, make sure the maximum data length is not over 4000 bytes.
VARCHAR( $n > 4000$ )	CLOB	Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
XML	XMLTYPE	The Oracle database server checks the syntax of the XMLTYPE data but SQL Anywhere does not.  Not available in UltraLite.

<sup>1</sup> Only applies to Oracle Database 11g or later.

### **i** Note

The LONG data types are deprecated in Oracle 8, 8i and 9i.

## Mapping to SQL Anywhere or UltraLite Remote Data Types

The following table identifies how Oracle consolidated data types are mapped to SQL Anywhere and UltraLite remote data types. For example, a column of type LONG on the consolidated database should be type LONG VARCHAR on the remote database.

Oracle data type	SQL Anywhere or UltraLite data type	Notes
BFILE	LONG BINARY	Download only.  Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.

Oracle data type	SQL Anywhere or UltraLite data type	Notes
BINARY_DOUBLE	DOUBLE	The special values INF, -INF and NAN of BINARY_FLOAT cannot be synchronized with SQL Anywhere or UltraLite. The precision of FLOAT and DOUBLE in Oracle is different from SQL Anywhere and UltraLite. The value of the data may change depending on the precision.
BINARY_FLOAT	REAL	The special values INF, -INF and NAN of BINARY_FLOAT cannot be synchronized with SQL Anywhere or UltraLite. The precision of FLOAT and DOUBLE in Oracle is different from SQL Anywhere and UltraLite. The value of the data may change depending on the precision.
BLOB	LONG BINARY	Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
CHAR( <i>n</i> byte)	VARCHAR( <i>n</i> )	There is no equivalence between SQL Anywhere CHAR and Oracle CHAR. SQL Anywhere CHAR is equivalent to VARCHAR. You should not use CHAR/NCHAR in a consolidated database column that is synchronized. If you must use non-SQL Anywhere CHAR, run the MobiLink server with the -b option.  SQL Anywhere or UltraLite values can be longer than Oracle values, so make sure values are not too big when uploading.
CLOB	LONG VARCHAR	Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.
DATE	TIMESTAMP	The year must be in the range 1-9999.
INTERVAL YEAR( <i>year_precision</i> ) TO MONTH		There is no corresponding data type in SQL Anywhere or UltraLite.
INTERVAL DAY( <i>day_precision</i> ) TO SECOND( <i>p</i> )		There is no corresponding data type in SQL Anywhere or UltraLite.
LONG	LONG VARCHAR	

Oracle data type	SQL Anywhere or UltraLite data type	Notes
LONG RAW	LONG BINARY	
NCHAR( <i>c</i> char)	NVARCHAR( <i>c</i> )	<p>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.</p> <p>SQL Anywhere or UltraLite values can be longer than Oracle values, so make sure values are not too big when uploading.</p>
NCLOB	LONG NVARCHAR	<p>Not available in UltraLite.</p> <p>Oracle values can be longer than SQL Anywhere or UltraLite values, so make sure values are not too big when downloading.</p>
NUMBER( <i>p,s</i> )	NUMBER( <i>p,s</i> )	<p>In SQL Anywhere NUMBER, <i>p</i> is between 1 and 127, and <i>s</i> is always less than or equal to <i>p</i>. In Oracle NUMBER, <i>p</i> ranges from 1 to 38, and <i>s</i> ranges from -84 to 127. To synchronize, the Oracle NUMBER scale must be between 0 and 38.</p>
NVARCHAR2( <i>c</i> char)	NVARCHAR( <i>c</i> )	<p>Not available in UltraLite.</p> <p>SQL Anywhere or UltraLite values can be longer than Oracle values, so make sure values are not too big when uploading.</p>
RAW	BINARY	<p>SQL Anywhere or UltraLite values can be longer than Oracle values, so make sure values are not too big when uploading.</p>
ROWID	VARCHAR(64)	<p>UROWID and ROWID are read-only and so are unlikely to be synchronized.</p>
SDO_GEOMETRY	ST_GEOMETRY	

Oracle data type	SQL Anywhere or UltraLite data type	Notes
TIMESTAMP( <i>p</i> <=6)	TIMESTAMP	When <i>p</i> <6, you may need to ensure SQL Anywhere or UltraLite values have the same precision. Otherwise, conflict detection may fail and/or duplicate rows may result. The year must be in the range 1-9999.
TIMESTAMP( <i>p</i> >6)		There is no corresponding data type in SQL Anywhere or UltraLite.
TIMESTAMP( <i>p</i> ) WITH LOCAL TIME ZONE		There is no corresponding data type in SQL Anywhere or UltraLite.
TIMESTAMP( <i>p</i> <=6) WITH TIME ZONE	TIMESTAMP WITH TIME ZONE	
TIMESTAMP( <i>p</i> ) WITH TIME ZONE		There is no corresponding data type in SQL Anywhere or UltraLite.
UROWID	VARCHAR(64)	UROWID and ROWID are read-only and so are unlikely to be synchronized.
VARCHAR2( <i>n</i> byte)	VARCHAR( <i>n</i> )	SQL Anywhere or UltraLite values can be longer than Oracle values, so make sure values are not too big when uploading.
XMLTYPE	XML, LONG VARCHAR or VARCHAR( <i>n</i> )	The Oracle database server checks the syntax of the XMLTYPE data. SQL Anywhere does not check the contents of the XML data, and treats the XML data as VARCHAR data.

## Related Information

[Oracle XMLTYPE Data Type \[page 176\]](#)

## 1.14.4.6 SAP HANA Data Mapping

SQL Anywhere and UltraLite remote data types can be mapped to SAP HANA consolidated data types and vice versa.

### 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.

SQL Anywhere or UltraLite data type	SAP HANA data type	Notes
BIT	TINYINT	
TINYINT	TINYINT	
SMALLINT	SMALLINT	
UNSIGNED SMALLINT	INTEGER	
INTEGER	INTEGER	
UNSIGNED INTEGER	BIGINT	
BIGINT	BIGINT	
UNSIGNED BIGINT	DECIMAL(20,0)	
DECIMAL(p,s)	DECIMAL(p,s)	
NUMERIC(p,s)	DECIMAL(p,s)	
FLOAT	FLOAT	The FLOAT data type should be avoided in remote databases with tables that download data from SAP HANA because HANA only supports DOUBLE values and they cannot be fully represented by FLOAT .
REAL	REAL	The REAL data type should be avoided in remote databases with tables that download data from SAP HANA because HANA only supports DOUBLE values and they cannot be fully represented by REAL.
DOUBLE	DOUBLE	

SQL Anywhere or UltraLite data type	SAP HANA data type	Notes
SMALLMONEY	DECIMAL(10,4)	
MONEY	DECIMAL(19,4)	
DATE	DATE	
TIME	TIME	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.
SMALLDATETIME	TIMESTAMP	
DATETIME	TIMESTAMP	
TIMESTAMP	TIMESTAMP	
TIMESTAMP WITH TIME ZONE	VARCHAR(34)	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:nn:ss.ssssss [+ -]hh:nn' and then applies it to the consolidated database. During download, the MobiLink server converts the data from a string to TIMESTAMP WITH TIME ZONE. Make sure the data in the consolidated database follows this format to avoid errors and synchronization failure.
CHAR( $n \leq 5000$ )	VARCHAR( $n$ )	
CHAR( $n > 5000$ )	CLOB	
VARCHAR( $n \leq 5000$ )	VARCHAR( $n$ )	
VARCHAR( $n > 5000$ )	CLOB	
LONG VARCHAR	CLOB	
NCHAR( $n \leq 5000$ )	NVARCHAR	
NCHAR( $n > 5000$ )	NCLOB	
NVARCHAR( $n \leq 5000$ )	NVARCHAR( $n$ )	

SQL Anywhere or UltraLite data type	SAP HANA data type	Notes
NVARCHAR( $n > 5000$ )	NCLOB	
LONG NVARCHAR	NCLOB	
BINARY( $n \leq 5000$ )	VARBINARY( $n$ )	
BINARY( $n > 5000$ )	BLOB	
VARBINARY( $n \leq 5000$ )	VARBINARY( $n$ )	
VARBINARY( $n > 5000$ )	BLOB	
LONG BINARY	BLOB	
VARBIT( $n \leq 5000$ )	VARCHAR( $n$ )	
VARBIT( $n > 5000$ )	CLOB	
LONG VARBIT	CLOB	
UNIQUEIDENTIFIER	VARCHAR(36)	
ST_GEOMETRY		

## 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.

SAP HANA data type	SQL Anywhere or UltraLite data type	Notes
TINYINT	TINYINT	
SMALLINT	SMALLINT	
INTEGER	INTEGER	
BIGINT	BIGINT	
SMALLDECIMAL	DECIMAL( $p,s$ )	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.

SAP HANA data type	SQL Anywhere or UltraLite data type	Notes
DECIMAL( <i>p,s</i> )	DECIMAL( <i>p,s</i> )	In SQL Anywhere, DECIMAL <i>p</i> is between 1 and 127 and <i>s</i> is always less than or equal to <i>p</i> . In SAP HANA, <i>p</i> ranges from 1 to 34 and <i>s</i> ranges from -6111 to 6176.
FLOAT	DOUBLE	SAP HANA promotes FLOAT and REAL to DOUBLE.
REAL	DOUBLE	SAP HANA promotes FLOAT and REAL to DOUBLE.
DOUBLE	DOUBLE	
DATE	DATE	
TIME	TIME	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.
SECONDDATE	TIMESTAMP	SQL Anywhere and UltraLite fractional seconds cannot be preserved when using an SAP HANA SECONDDATE data type, which has no fractional seconds. To avoid problems, do not use fractional seconds.
TIMESTAMP	TIMESTAMP	
VARCHAR( <i>n</i> )	VARCHAR( <i>n</i> )	
NVARCHAR( <i>n</i> )	NVARCHAR( <i>n</i> )	
ALPHANUM( <i>n</i> )	VARCHAR( <i>n</i> )	
VARBINARY( <i>n</i> )	VARBINARY( <i>n</i> )	
CLOB	LONG VARCHAR	
NCLOB	LONG NVARCHAR	
BLOB	LONG BINARY	
ST_GEOMETRY	ST_GEOMETRY	



## 1.14.4.7 SAP IQ Enterprise Data Mapping

SQL Anywhere and UltraLite remote data types can be mapped to SAP IQ Enterprise consolidated data types and vice versa.

### Mapping to SAP IQ Consolidated Data Types

The following table identifies how SQL Anywhere and UltraLite remote data types are mapped to SAP 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.

SQL Anywhere or UltraLite data type	SAP IQ	Notes
BIGINT	BIGINT	
BIT	BIT	
BINARY( <i>n</i> )	BINARY( <i>n</i> )	
CHAR( <i>n</i> )	CHAR( <i>n</i> )	There are some restrictions on CHAR and VARCHAR columns over 255 bytes. For more information, see the SAP IQ documentation.
DATE	DATE	
DATETIME	DATETIME	
DECIMAL( <i>p,s</i> )	DECIMAL( <i>p,s</i> )	
DOUBLE	DOUBLE	
FLOAT( <i>p</i> )	FLOAT( <i>p</i> )	
INT	INT	
LONG BINARY / IMAGE	LONG BINARY / IMAGE	
LONG NVARCHAR / NTEXT		This data type is not available in SAP IQ.
LONG VARBIT	LONG VARCHAR	
LONG VARCHAR / TEXT	TEXT	
MONEY	MONEY	
NCHAR( <i>n</i> )		This data type is not available in SAP IQ.
NVARCHAR( <i>n</i> )		This data type is not available in SAP IQ.

SQL Anywhere or UltraLite data type	SAP IQ	Notes
NUMERIC(p,s)	NUMERIC(p,s)	
SMALLDATETIME	SMALLDATETIME	
SMALLMONEY	SMALLMONEY	
ST_GEOMETRY		This data type is not available in SAP IQ.
TIME	TIME	
TIMESTAMP	TIMESTAMP	
TIMESTAMP WITH TIME ZONE	VARCHAR(34)	
TINYINT	TINYINT	
UNIQUEIDENTIFIER	UNIQUEIDENTIFIER	
UNSIGNED BIGINT	UNSIGNED BIGINT	
UNSIGNED INT	UNSIGNED INT	
UNSIGNED SMALLINT	SMALLINT	
UNSIGNED TINYINT	TINYINT	
VARBINARY(n)	VARBINARY(n)	
VARBIT(n)	VARCHAR(n)	
VARCHAR(n)	VARCHAR(n)	There are some restrictions on CHAR and VARCHAR columns over 255 bytes. For more information, see the SAP IQ documentation.
XML	LONG BINARY / IMAGE	

## Mapping to SQL Anywhere or UltraLite Remote Data Types

The following table identifies how SAP 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.

SAP IQ	SQL Anywhere or UltraLite data type	Notes
BIGINT	BIGINT	
BINARY(n)	BINARY(n)	

SAP IQ	SQL Anywhere or UltraLite data type	Notes
BIT	BIT	
CHAR(n)	VARCHAR(n)	
DATE	DATE	
DATETIME	DATETIME	
DECIMAL(p,s)	DECIMAL(p,s)	
DOUBLE	DOUBLE	
FLOAT(p)	FLOAT(p)	
INT	INT	
LONG BINARY / IMAGE	LONG BINARY / IMAGE	
LONG VARCHAR / TEXT	LONG VARCHAR / TEXT	
MONEY	MONEY	
NUMERIC(p,s)	NUMERIC(p,s)	
REAL	REAL	
SMALLDATETIME	SMALLDATETIME	
SMALLINT	SMALLINT	
SMALLMONEY	SMALLMONEY	
TIME	TIME	
TIMESTAMP	TIMESTAMP	
TINYINT	TINYINT	
UNIQUEIDENTIFIER	UNIQUEIDENTIFIER	
UNSIGNED BIGINT	UNSIGNED BIGINT	
UNSIGNED INT	UNSIGNED INT	
VARBINARY(n)	VARBINARY(n)	
VARCHAR(n)	VARCHAR(n)	

## 1.14.5 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.

### i 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.

#### In this section:

[Character Set Conversion During Synchronization \[page 708\]](#)

During synchronization, characters may need to be converted from one character set to another.

## Related Information

[International Languages and Character Sets](#)

[UltraLite Character Sets](#)

[MobiLink Consolidated Databases \[page 152\]](#)

[Recommended ODBC Drivers For MobiLink](#)

### 1.14.5.1 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.

## Example

- UltraLite applications on Microsoft Windows Mobile devices use the Unicode character set. When you synchronize a Microsoft 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 Microsoft 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.

### In this section:

#### [ODBC Driver Character Set Conversion \[page 710\]](#)

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.

## 1.14.5.1.1 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 SAP 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.

## 1.14.6 ODBC Drivers for MobiLink

The MobiLink server can work with a variety of consolidated databases and ODBC drivers. Some drivers, though compatible for use with MobiLink, may have functional restrictions associated with their use.

### In this section:

[SQL Anywhere 17 - Oracle ODBC Driver \[page 710\]](#)

The SQL Anywhere 17 - Oracle ODBC driver is custom-tailored for use with SQL Anywhere software. This driver does not work with third-party software.

## Related Information

[Supported Platforms](#)

### 1.14.6.1 SQL Anywhere 17 - Oracle ODBC Driver

The SQL Anywhere 17 - Oracle ODBC driver is custom-tailored for use with SQL Anywhere software. This driver does not work with third-party software.

If you use Oracle with MobiLink or remote data access, then install an Oracle client on the same computer as this Oracle driver.

The Oracle driver can be configured using the ODBC Data Source Administrator (Microsoft Windows), the Data Source utility (dbdsn), or by editing the `.odbc.ini` file (UNIX and Linux).

The following table provides the configuration options for the Oracle driver.

Microsoft Windows ODBC Data Source Administrator	Configuration for dbdsn command line or .odbc.ini file	Description
Data source name	For dbdsn, use the <code>-w</code> option and specify the data source name.	A name to identify your data source.
User ID	For dbdsn, use the <code>UserID</code> connection parameter in the connection string. Short form <code>UID</code> .	The default logon ID that the application uses to connect to your Oracle database. If you leave this field blank, you are prompted for the information when you connect.
Password	For dbdsn, use the <code>Password</code> connection parameter in the connection string. Short form <code>PWD</code> .	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.
TNS service name	For dbdsn, use the <code>ServiceName</code> connection parameter in the connection string. Short form <code>SN</code> .	The TNS Service Name that is stored in <code>network/admin/tnsnames.ora</code> under your Oracle installation directory.
Encode password	For dbdsn, use the <code>-pet</code> option and specify the plain text password in the connection string by using the <code>Password</code> or <code>PWD</code> connection parameter.	Select this option to store the password in an encoded form in the data source. When using the ODBC Data Source Administrator, re-enter the password to change an existing encoding option.
Procedure returns results or uses VARRAY parameters	For dbdsn, use the <code>ProcResults</code> connection parameter in the connection string. Short form <code>PROC</code> .	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 check box.  If no stored procedures use VARRAYs and none of them returns a result set, clear this check box to improve performance.
Array size	For dbdsn, use the <code>ArraySize</code> connection parameter in the connection string. Short form <code>SIZE</code> .	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.

Microsoft Windows ODBC Data Source Administrator	Configuration for dbdsn command line or .odbc.ini file	Description
Enable Microsoft distributed transactions	For dbdsn, use the <i>EnableMSDTC</i> connection parameter in the connection string. Short form <i>EDTC</i> .  Not supported for UNIX and Linux.	Select this check box 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.
Fetch array size (rows)	For dbdsn, use the <i>FetchArraySize</i> connection parameter in the connection string.	The number of rows fetched from the Oracle database. The default value is 20. Increasing the value reduces the number of round trips on the network and increases performance, but also increases ODBC driver memory use.  The ideal setting varies with each application. To calculate the ideal setting, divide the network packet size (in bytes) by the size of the rows that you are fetching (in bytes). After you calculate this number, leave space for packet overhead. For example, if your Network Packet size is 1024 bytes and the row size is 8 bytes, then divide 1024 by 8, which equals 128. The ideal setting for this option is smaller than 128 because the number of rows times the row size must be slightly smaller than the network packet size.

#### In this section:

##### [Creating an ODBC Data Source for the Oracle Driver in Microsoft Windows \[page 713\]](#)

Use this procedure to create an ODBC data source for the Oracle driver in Microsoft Windows.

##### [UNIX/Linux Configuration \[page 713\]](#)

On UNIX and Linux, 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).

##### [Creating an ODBC Data Source for the Oracle Driver \(dbdsn Utility\) \[page 714\]](#)

To create an Oracle DSN with the dbdsn utility, use the following syntax.



## 1.14.6.1.1 Creating an ODBC Data Source for the Oracle Driver in Microsoft Windows

Use this procedure to create an ODBC data source for the Oracle driver in Microsoft Windows.

### Procedure

1. Open the ODBC Administrator:
  - Click **Start** > **Programs** > **SQL Anywhere 17** > **Administration Tools** > **Open ODBC Data Source Administrator**.

The *ODBC Data Source Administrator* appears.
2. Click *Add*.
3. Choose *SQL Anywhere 17 - Oracle* and click *Finish*.
4. Specify the configuration options you need.
5. Click *Test Connection*, and then click *OK*.

### Results

The ODBC data source for the Oracle driver is created.

### Next Steps

Use the ODBC data source to connect.

## 1.14.6.1.2 UNIX/Linux Configuration

On UNIX and Linux, if you are setting up the driver in an ODBC system information file (typically called `.odbc.ini`), the section for this driver should appear as follows (with appropriate values entered for each field).

```
[sample_dsn_using_the_ias_odbc_driver_for_oracle]
Driver=full-path/libdboraodbc17_r.so
UserID=user-id
Password=password
ServiceName=TNS-service-name
ProcResults=[yes|no]
ArraySize=bytes
```

### 1.14.6.1.3 Creating an ODBC Data Source for the Oracle Driver (dbdsn Utility)

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 the documentation for the SQL Anywhere 17 - Oracle ODBC driver.

For example:

```
dbdsn -w MyOracleDSN -or -pet u -c  
"Userid=dba;Password=passwd;ServiceName=abcd;ArraySize=100000;ProcResults=y"
```

## Related Information



[Recommended ODBC Drivers For MobiLink](#) 

# Important Disclaimers and Legal Information

## Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.

About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

## Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

## Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.

The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

## Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

## Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

© 2022 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see <https://www.sap.com/about/legal/trademark.html> for additional trademark information and notices.