



MobiLink™

Client Administration

Copyright © 2010 iAnywhere Solutions, Inc. Portions copyright © 2010 Sybase, Inc. All rights reserved.

This documentation is provided AS IS, without warranty or liability of any kind (unless provided by a separate written agreement between you and iAnywhere).

You may use, print, reproduce, and distribute this documentation (in whole or in part) subject to the following conditions: 1) you must retain this and all other proprietary notices, on all copies of the documentation or portions thereof, 2) you may not modify the documentation, 3) you may not do anything to indicate that you or anyone other than iAnywhere is the author or source of the documentation.

iAnywhere®, Sybase®, and the marks listed at <http://www.sybase.com/detail?id=1011207> are trademarks of Sybase, Inc. or its subsidiaries. ® indicates registration in the United States of America.

All other company and product names mentioned may be trademarks of the respective companies with which they are associated.

Contents

About this book	v
About the SQL Anywhere documentation	v
Introduction to MobiLink clients	1
MobiLink clients	1
MobiLink users	4
MobiLink client utilities	18
MobiLink client network protocol options	23
Schema changes in remote clients	58
SQL Anywhere clients for MobiLink	63
SQL Anywhere clients	63
MobiLink SQL Anywhere client utility (dbmlsync)	93
MobiLink SQL Anywhere client extended options	126
MobiLink SQL statements	156
MobiLink synchronization profiles	157
Event hooks for SQL Anywhere clients	178
Dbmlsync C++ API reference	231
Dbmlsync .NET API reference	253
Dbmlsync integration component (removed)	276
DBTools interface for dbmlsync	304
Scripted upload	310
Index	331

About this book

This book describes how to set up, configure, and synchronize MobiLink clients. MobiLink clients can be SQL Anywhere or UltraLite databases. This book also describes the Dbmlsync API, which allows you to integrate synchronization seamlessly into your C++ or .NET client applications.

About the SQL Anywhere documentation

The complete SQL Anywhere documentation is available in four formats:

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

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

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

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

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

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

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

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

Documentation conventions

This section lists the conventions used in this documentation.

Operating systems

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

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

- **Windows** The Microsoft Windows family includes platforms that are used primarily on server, desktop, and laptop computers, as well as platforms used on mobile devices. Unless otherwise specified, when the documentation refers to Windows, it refers to all supported Windows-based platforms, including Windows Mobile.

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

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

For the complete list of platforms supported by SQL Anywhere, see [“Supported platforms” \[SQL Anywhere 12 - Introduction\]](#).

Directory and file names

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

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

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

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

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

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

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

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

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

For example, on Windows, the network database server is *dsrv12.exe*. On Unix, it is *dsrv12*.

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

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

For more information about the default location of *install-dir*, see [“SQLANY12 environment variable” \[SQL Anywhere Server - Database Administration\]](#).

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

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

For more information about the default location of *samples-dir*, see [“SQLANYSAMP12 environment variable” \[SQL Anywhere Server - Database Administration\]](#).

Command prompts and command shell syntax

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

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

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

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

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

Where parentheses cause syntax problems, substitute curly braces:

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

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

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

- **Semicolons** On Unix, semicolons should be enclosed in quotes.
- **Quotes** If you must specify quotes in a parameter value, the quotes may conflict with the traditional use of quotes to enclose the parameter. For example, to specify an encryption key whose value contains double-quotes, you might have to enclose the key in quotes and then escape the embedded quote:

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

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

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

Contacting the documentation team

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

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

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

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

Finding out more and requesting technical support

Newsgroups

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

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

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

The newsgroups include the following:

- [sybase.public.sqlanywhere.general](#)
- [sybase.public.sqlanywhere.linux](#)
- [sybase.public.sqlanywhere.mobilink](#)
- [sybase.public.sqlanywhere.product_futures_discussion](#)
- [sybase.public.sqlanywhere.replication](#)
- [sybase.public.sqlanywhere.ultralite](#)
- [ianywhere.public.sqlanywhere.qanywhere](#)

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

Newsgroup disclaimer

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

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

Developer Centers

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

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

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

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

Introduction to MobiLink clients

This section introduces the clients you can use for MobiLink synchronization, and provides information common to all types of MobiLink client.

MobiLink clients

The following sections contain information about MobiLink clients.

SQL Anywhere clients

To use a SQL Anywhere database as a MobiLink client, you add synchronization objects to the database. The objects you need to add are publications, MobiLink users, and subscriptions that connect publications to users. See:

- [“Creating a remote database” on page 63](#)
- [“Publishing data” on page 67](#)
- [“Creating MobiLink users” on page 74](#)
- [“Creating synchronization subscriptions” on page 77](#)

Synchronization can be initiated using the Dbmlsync API, the SQL SYNCHRONIZE statement or the dbmlsync command line utility. Most synchronizations use data read from the database transaction log, however, a transaction log file is not required for scripted-upload synchronization and synchronization of download-only publications.

See [“Initiating synchronization” on page 79](#).

For more information about SQL Anywhere clients, see [“SQL Anywhere clients” on page 63](#).

For details of dbmlsync command line options, see [“MobiLink SQL Anywhere client utility \(dbmlsync\)” on page 93](#).

For information about customizing synchronization, see [“Customizing dbmlsync synchronization” on page 89](#).

UltraLite clients

UltraLite applications are automatically MobiLink-enabled whenever the application includes a call to the appropriate synchronization function.

The UltraLite application and libraries handle the synchronization actions at the application end. You can write your UltraLite application with little regard to synchronization. The UltraLite runtime keeps track of changes made since the previous synchronization.

When using TCP/IP, HTTP, HTTPS, or Microsoft ActiveSync, synchronization is initiated from your application by a single call to a synchronization function.

See also

- [“UltraLite clients” on page 1](#)
- [“Using ActiveSync with UltraLite on Windows Mobile” \[UltraLite - Database Management and Reference\]](#)
- [“UltraLite synchronization parameters and network protocol options” \[UltraLite - Database Management and Reference\]](#)
- [“UltraLite - Database Management and Reference”](#)
- [“UltraLiteJ clients” on page 2](#)

UltraLiteJ clients

UltraLiteJ provides Java applications with a MobiLink synchronization client, together with change-tracking and state tracking to ensure robust synchronization. UltraLiteJ applications are automatically MobiLink-enabled whenever the application includes a call to the appropriate synchronization function.

The UltraLiteJ application and libraries handle the synchronization actions at the application end. You can write your UltraLiteJ application with little regard to synchronization. The UltraLiteJ runtime keeps track of changes made since the previous synchronization.

When using HTTP or HTTPS, synchronization is initiated from your application by a single call to a synchronization function.

The MobiLink File Transfer utility (mlfiletransfer) is not available for UltraLiteJ clients.

See also

- [“Data synchronization” \[UltraLiteJ\]](#)
- [“Synchronizing with MobiLink” \[UltraLiteJ\]](#)
- [“Using UltraLiteJ as a MobiLink client” \[UltraLiteJ\]](#)
- [“Synchronizing the CustDB application” \[UltraLiteJ\]](#)
- [“Network protocol options for UltraLiteJ synchronization streams” \[UltraLiteJ\]](#)
- [“UltraLite - Database Management and Reference”](#)
- [“UltraLite clients” on page 1](#)

Specifying the network protocol for clients

The MobiLink server uses the `-x` command line option to specify the network protocol or protocols for synchronization clients to connect to the MobiLink server. The network protocol you choose must match the synchronization protocol used by the client.

The syntax for the `mlsrv12` command line option is:

```
mlsrv12 -c "connection-string" -x protocol( options )
```

In the following example, the TCP/IP protocol is selected with no additional protocol options.

```
mlsrv12 -c "dsn=SQL Anywhere 12 Demo" -x tcpip
```

You can configure your protocol using options of the form:

(*keyword=value;...*)

For example:

```
mlsrv12 -c "dsn=SQL Anywhere 12 Demo" -x tcpip(
    host=localhost;port=2439)
```

See also

Complete details about MobiLink network protocols and protocol options can be found in the following locations:

To find...	See...
How to set network options for the MobiLink server	“-x mlsrv12 option” [<i>MobiLink - Server Administration</i>]
All the network protocol options available to MobiLink client applications	“MobiLink client network protocol option summary” on page 23
How to set options for SQL Anywhere clients	“CommunicationAddress (adr) extended option” on page 128 “CommunicationType (ctp) extended option” on page 129
How to set options for UltraLite clients	“Stream Parameters synchronization parameter” [<i>UltraLite - Database Management and Reference</i>] “Stream Type synchronization parameter” [<i>UltraLite - Database Management and Reference</i>] “UltraLite Synchronization utility (ulsync)” [<i>UltraLite - Database Management and Reference</i>]

System tables in MobiLink

MobiLink server system tables

When you set up a database for use as a consolidated database, MobiLink system tables are created that are required by the MobiLink server.

See [“MobiLink server system tables”](#) [*MobiLink - Server Administration*].

UltraLite system tables

See “UltraLite system tables” [[UltraLite - Database Management and Reference](#)].

SQL Anywhere system tables

SQL Anywhere system tables cannot be accessed directly, but are accessed via system views. See “System views” [[SQL Anywhere Server - SQL Reference](#)].

The following SQL Anywhere system views are of particular interest to MobiLink users:

- “SYSSYNC system view” [[SQL Anywhere Server - SQL Reference](#)]
- “SYSPUBLICATION system view” [[SQL Anywhere Server - SQL Reference](#)]
- “SYSSUBSCRIPTION system view” [[SQL Anywhere Server - SQL Reference](#)]
- “SYSSYNCSRIPT system view” [[SQL Anywhere Server - SQL Reference](#)]
- “SYSSYNCPROFILE system view” [[SQL Anywhere Server - SQL Reference](#)]
- “SYSARTICLE system view” [[SQL Anywhere Server - SQL Reference](#)]
- “SYSARTICLECOL system view” [[SQL Anywhere Server - SQL Reference](#)]

SQL Anywhere also provides consolidated views that query system views to provide information that you might need. See “Consolidated views” [[SQL Anywhere Server - SQL Reference](#)].

MobiLink users

Introduction to MobiLink users

A **MobiLink user**, also called a **synchronization user**, is the name you use to authenticate when you connect to the MobiLink server.

For a user to be part of a synchronization system:

- A MobiLink user name must be created on the remote database.
- The MobiLink user name must be registered with the MobiLink server.

MobiLink user names and passwords are not the same as database user names and passwords. MobiLink user names are used to authenticate the connection from the remote database to the MobiLink server.

MobiLink user names are always case sensitive unless you are using custom authentication scripts, so the user name provided by the remote database has to exactly match the case of the user name that was registered in the consolidated database. Keep the following in mind when defining MobiLink user names:

- **In a MobiLink synchronization system, no two user names can be the same except for case** For example, you can have user name **aA** or user name **Aa**, but not both.

- **For any given user name, you must always use the same case once you start using it** For example, if you add a user as **Aa** and synchronize with it, you must continue to synchronize using **Aa**. Using **aA** will fail.

When using custom authentication scripts, the script determines the case sensitivity of user names.

You can also use user names to control the behavior of the MobiLink server. You do so using the **username** parameter in synchronization scripts. See [“Using remote IDs and MobiLink user names in scripts” on page 10](#).

The MobiLink user name is stored in the name column of the ml_user MobiLink system table in the consolidated database.

The MobiLink user name does not have to be unique within your synchronization system. If security is not an issue, you can even assign the same MobiLink user name to every remote database.

UltraLite user authentication

Although UltraLite and MobiLink user authentication schemes are separate, you may want to share the values of UltraLite user IDs with MobiLink user names for simplicity. This only works when the UltraLite application is used by a single user.

See [“UltraLite user authentication” \[UltraLite - Database Management and Reference\]](#).

Creating and registering MobiLink users

You create a MobiLink user in the remote database and register it in the consolidated database.

Creating MobiLink users in the remote database

To add users to the remote database, you have the following options:

- For SQL Anywhere remote databases, use Sybase Central or the CREATE SYNCHRONIZATION USER statement.
See [“Creating MobiLink users” on page 74](#).
- For UltraLite remote databases, you set the User Name and Password synchronization parameters.
See [“User Name synchronization parameter” \[UltraLite - Database Management and Reference\]](#) and [“Password synchronization parameter” \[UltraLite - Database Management and Reference\]](#).

Adding MobiLink user names to the consolidated database

Once user names are created in the remote database, you can use any of the following methods to register the user names in the consolidated database:

- Use the mluser utility.
See [“MobiLink user authentication utility \(mluser\)” \[MobiLink - Server Administration\]](#).
- Use Sybase Central.

- Implement a script for the `authenticate_user` or `authenticate_user_hashed` events. When either of these scripts are invoked, the MobiLink server automatically adds users that successfully authenticate.
See “[authenticate_user connection event](#)” [*MobiLink - Server Administration*] or “[authenticate_user_hashed connection event](#)” [*MobiLink - Server Administration*].
- Specify the `-zu+` command line option with `mlsrv12`. In this case, any existing MobiLink users that have not been added to the consolidated database are added when they first synchronize. This option is useful during development but is not recommended for deployed applications.
See “[-zu mlsrv12 option](#)” [*MobiLink - Server Administration*].

Providing initial passwords for users

The password for each user is stored with the user name in the `ml_user` table. You can use Sybase Central or the `mluser` command line utility to provide initial passwords.

Sybase Central is a convenient way of adding individual users and passwords. The `mluser` utility is useful for batch additions.

If you create a user without a password, MobiLink does not authenticate the user and a password is not required to connect and synchronize.

To provide an initial MobiLink password for a user (Sybase Central)

1. From the **View** menu, choose **Folders**.
2. Open your MobiLink project and expand **Consolidated Databases** using the MobiLink 12 plug-in.
3. Expand the name of your consolidated database.
4. Click **Users**.
5. Choose **File » New » User**.
6. Follow the instructions in the **Create User Wizard**.

To provide initial MobiLink passwords (command line)

1. Create a file with a single user name and password on each line, separated by white space.
2. Open a command prompt and run the `mluser` command line utility. For example:

```
mluser -c "dsn=my_dsn" -f password-file
```

In this command line, the `-c` option specifies an ODBC connection to the consolidated database. The `-f` option specifies the file containing the user names and passwords.

See “[MobiLink user authentication utility \(mluser\)](#)” [*MobiLink - Server Administration*].

Synchronizations from new users

Ordinarily, each MobiLink client must provide a valid MobiLink user name and password to connect to a MobiLink server.

Setting the `-zu+` option when you start the MobiLink server allows the server to accept and respond to synchronization requests from unregistered users. When a request is received from a user not listed in the `ml_user` table, the request is serviced and the user is added to the `ml_user` table.

When you use `-zu+`, if a MobiLink client synchronizes with a user name that is not in the current `ml_user` table, MobiLink, by default, takes the following actions:

- **New user, no password** If the user supplied no password, then the user name is added to the `ml_user` table with a null password. This user is allowed to synchronize without a password.
- **New user, password** If the user supplies a password, then the user name and password are both added to the `ml_user` table and the new user name becomes a recognized name in your MobiLink system. In future, this user must specify the same password to synchronize.
- **New user, new password** A new user may provide information in the new password field, or in the password field. In either case, the new password setting overrides the old password setting, and the new user is added to the MobiLink system using the new password. In future, this user must specify the same password to synchronize.

See “`-zu mlsrv12` option” [*MobiLink - Server Administration*].

Preventing synchronization by unknown users

By default, the MobiLink server only recognizes users who are registered in the `ml_user` table. This default provides two benefits. First, it reduces the risk of unauthorized access to the MobiLink server. Second, it prevents authorized users from accidentally connecting using an incorrect or misspelled user name. Such accidents should be avoided because they can cause the MobiLink system to behave in unpredictable ways.

Prompting end users to enter passwords

Each end user must supply a MobiLink user name and password each time they synchronize from a MobiLink client, unless you choose to disable user authentication on your MobiLink server.

To prompt your end users to enter their MobiLink passwords

- The mechanism for supplying the user name and password is different for UltraLite and SQL Anywhere clients.
 - **UltraLite** When synchronizing, the UltraLite client must supply a valid value in the password field of the synchronization structure. For built-in MobiLink synchronization, a valid password is one that matches the value in the `ml_user` MobiLink system table.

Your application should prompt the end user to enter their MobiLink user name and password before synchronizing.

See [“UltraLite synchronization parameters and network protocol options”](#) [*UltraLite - Database Management and Reference*].

- **SQL Anywhere** Users can supply a valid password on the dbmlsync command line using the -mp option, or store it in the database with the synchronization subscription using the MobilinkPwd extended option. Otherwise, they are prompt to specify a password in the dbmlsync connection window. The latter method is more secure than specifying the password on the command line because command lines are visible to other processes running on the same computer.

If authentication fails, the user is prompted to re-enter the user name and password.

See:

- [“-c dbmlsync option”](#) on page 101
- [“-mp dbmlsync option”](#) on page 110
- [“MobiLinkPwd \(mp\) extended option”](#) on page 140

Changing passwords

MobiLink provides a mechanism for end users to change their password. The interface differs between UltraLite and SQL Anywhere clients.

To prompt your end users to enter MobiLink passwords

- The mechanism for supplying the user name and password is different for UltraLite and SQL Anywhere clients.
- **SQL Anywhere** Supply a valid existing password together with the new password on the dbmlsync command line, or in the dbmlsync connection window if you do not supply command line parameters.

See [“-mp dbmlsync option”](#) on page 110 and [“-mn dbmlsync option”](#) on page 109.

- **UltraLite** When synchronizing, the application must supply the existing password in the password field of the synchronization structure and the new password in the new_password field.

See [“Password synchronization parameter”](#) [*UltraLite - Database Management and Reference*] and [“New Password synchronization parameter”](#) [*UltraLite - Database Management and Reference*].

An initial password can be set in the consolidated server or on the first synchronization attempt. See [“Providing initial passwords for users”](#) on page 6 and [“Synchronizations from new users”](#) on page 7.

Once a password is assigned, you cannot reset the password to an empty string from the client side.

Remote IDs

The **remote ID** uniquely identifies a remote database in a MobiLink synchronization system.

When a SQL Anywhere or UltraLite database is created, it is given a remote ID of null. When the database synchronizes with MobiLink, the MobiLink client checks for a null remote ID. If it finds a null remote ID, it assigns a GUID as the remote ID. Once set, the database maintains the same remote ID unless it is manually changed (changing a remote ID manually is not recommended).

For SQL Anywhere remote databases, the MobiLink server tracks synchronization progress by remote ID and subscription. For UltraLite databases, the MobiLink server tracks synchronization progress by remote ID and publication. This information is stored in the ml_subscription system table. The remote ID is also recorded in the MobiLink server log for each synchronization.

Remote IDs must be unique

Each remote database must be uniquely identified by its remote ID. The built-in GUID remote ID accomplishes this. For an alternative, more human-readable identifier to represent a remote database in your scripts and business logic, consider using the MobiLink username and/or a unique authentication parameter. If you must assign your own remote IDs, then you must ensure that each remote database is assigned a unique remote ID.

If the same remote ID is used in two or more concurrent synchronizations, it can potentially cause corruption and/or data loss, depending on your synchronization scripts and business logic. Concurrent synchronizations with the same remote ID can happen for either of the following reasons:

- A remote database has been assigned a duplicate remote ID. In this case the duplicate remote ID must be set to a unique remote ID.
- A network error disconnects the client, which synchronizes again immediately. It is possible for both the original and the new synchronization to be processed at the same time.

In either case, the MobiLink server automatically tries to prevent corruption or data loss. To do this, MobiLink server will typically cancel all but one of the concurrent synchronizations are cancelled with an error.

A caution about reusing remote IDs

If you set the remote ID manually and you subsequently recreate the remote database, you must either give the recreated remote database a different name from the old one or use the ml_reset_sync_state stored procedure to reset the state information in the consolidated database for the remote database. For more detailed information, see “[ml_reset_sync_state system procedure](#)” [*MobiLink - Server Administration*].

See also

- “[Introduction to MobiLink users](#)” on page 4
- “[Authentication parameters](#)” [*MobiLink - Server Administration*]
- “[ml_reset_sync_state system procedure](#)” [*MobiLink - Server Administration*]

Setting the MobiLink remote ID

The remote ID is created as a GUID, but you can change it to a more meaningful name. For both SQL Anywhere and UltraLite databases, the remote ID is stored in the database as a property called `ml_remote_id`.

For SQL Anywhere clients, see “[Setting remote IDs](#)” on page 65.

For UltraLite clients, see “[UltraLite ml_remote_id option](#)” [*UltraLite - Database Management and Reference*].

If you set the remote ID manually and you subsequently recreate the remote database, you must either give the recreated remote database a different name from the old one or use the `ml_reset_sync_state` stored procedure to reset the state information in the consolidated database for the remote database. “[ml_reset_sync_state system procedure](#)” [*MobiLink - Server Administration*]

When deploying a starter database to multiple locations, it is safest to deploy databases that have a null remote ID. If you have synchronized the databases to prepopulate them, you can set the remote ID back to null before deployment. This method ensures that the remote ID is unique because the first time the remote database synchronizes, a unique remote ID is assigned. Alternatively, the remote ID can be set as a remote setup step, but it must be unique.

Example

To simplify administrative duties when defining a MobiLink setup where you have one user per remote, you might want to use the same number for all three MobiLink identifiers on each remote database. For example, in a SQL Anywhere remote database you can set them as follows:

```
-- Set the MobiLink user name:
CREATE SYNCHRONIZATION USER "1" ... ;

-- Set the partition number for DEFAULT GLOBAL AUTOINCREMENT:
SET OPTION PUBLIC.GLOBAL_DATABASE_ID = '1';

-- Set the MobiLink remote ID:
SET OPTION PUBLIC.ml_remote_id = '1';
```

Using remote IDs and MobiLink user names in scripts

The MobiLink user identifies a person and is used for authentication. The remote ID uniquely identifies a MobiLink remote database.

In many synchronization scripts, you have the option of identifying the remote database by the remote ID (with the named parameter `s.remote_id`) or by the MobiLink user name (with `s.username`). Using the remote ID has some advantages, especially in UltraLite.

When deployments have a one-to-one relationship between a remote database and a MobiLink user, you can ignore the remote ID. In this case MobiLink event scripts can reference the username parameter, which is the MobiLink user name used for authentication.

If a MobiLink user wants to synchronize data in different databases but each remote has the same data, the synchronization scripts can reference the MobiLink user name. But if the MobiLink user wants to

synchronize different sets of data in different databases, the synchronization scripts should reference the remote ID.

In UltraLite databases, the same database can also be synchronized by different users, even if the previous upload state is unknown, because the MobiLink server tracks synchronization progress by remote ID. In this case, you can no longer reference the MobiLink user name in download scripts for timestamp-based downloads, because some rows for each of the other users may be missed and never downloaded. To prevent this, you need to implement a mapping table in the consolidated database with one row for each user using the same remote database. You can make sure that all data for all users is being downloaded with a join of the consolidated table and mapping table that is based on the remote ID for the current synchronization.

You can also use different script versions to synchronize different data to different remote databases. See [“Script versions” \[MobiLink - Server Administration\]](#).

Choosing a user authentication mechanism

User authentication is one part of a security system for protecting your data.

MobiLink provides you with a choice of user authentication mechanisms. You do not have to use a single installation-wide mechanism; MobiLink lets you use different authentication mechanisms for different script versions within the installation for flexibility.

- **No MobiLink user authentication** If your data is such that you do not need password protection, you can choose not to use any user authentication in your installation. In this case, the MobiLink user name must still be included in the ml_user table, but the hashed_password column is null.
- **Built-in MobiLink user authentication** MobiLink uses the user names and passwords stored in the ml_user MobiLink system table to perform authentication.

The built-in mechanism is described in the following sections.

- **Custom authentication** You can use the MobiLink script authenticate_user to replace the built-in MobiLink user authentication system with one of your own. For example, depending on your consolidated database management system, you may be able to use the database user authentication instead of the MobiLink system.

See [“Custom user authentication” on page 14](#).

For information about other security-related features of MobiLink and its related products, see:

- [“Encrypting MobiLink client/server communications” \[SQL Anywhere Server - Database Administration\]](#)
- UltraLite clients: [“Securing UltraLite databases” \[UltraLite - Database Management and Reference\]](#)
- SQL Anywhere clients: [“Keeping your data secure” \[SQL Anywhere Server - Database Administration\]](#)

User authentication architecture

The MobiLink user authentication system relies on user names and passwords. You can choose either to let the MobiLink server validate the user name and password using a built-in mechanism, or you can implement your own custom user authentication mechanism.

In the built-in authentication system, both the user name and the password are stored in the `ml_user` MobiLink system table in the consolidated database. The password is stored in hashed form so that applications other than the MobiLink server cannot read the `ml_user` table and reconstruct the original form of the password. You add user names and passwords to the consolidated database using Sybase Central, using the `mluser` utility, or by specifying `-zu+` when you start the MobiLink server.

See [“Creating and registering MobiLink users” on page 5](#).

When a MobiLink client connects to a MobiLink server, it provides the following values:

- **user name** The MobiLink user name. Mandatory. To synchronize, the user name must be stored in the `ml_user` system table, or you must start the MobiLink server with the `-zu+` option to add new users to the `ml_user` table.
- **password** The MobiLink password. Optional only if the user is unknown or if the corresponding password in the `ml_user` MobiLink system table is null.
- **new password** A new MobiLink password. Optional. MobiLink users can change their password by setting this value.

Custom authentication

Optionally, you can substitute your own user authentication mechanism.

See [“Custom user authentication” on page 14](#).

Authentication process

The following is an explanation of the order of events that occur during authentication.

1. A remote application initiates a synchronization request using a remote ID, a MobiLink user name, and optionally a password and new password. The MobiLink server starts a new transaction and triggers the `begin_connection_autocommit` event and `begin_connection` event.
2. MobiLink verifies that the remote ID is not currently synchronizing and presets the `authentication_status` to be 4000.
3. If you have defined an `authenticate_user` script, then the following occurs:
 - a. If the `authenticate_user` script is written in SQL, then this script is called with the preset `authentication_status` of 4000, the MobiLink user name you provided, and optionally the password and new password.

If the `authenticate_user` script is written in Java or .NET and returns a SQL statement, then this SQL statement is called with the preset `authentication_status` of 4000, the MobiLink user name you provided, and optionally the password and new password.

- b. If the `authenticate_user` script throws an exception or an error occurs in executing the script, the synchronization process stops.

The `authenticate_user` script or the returned SQL statement must be a call to a stored procedure taking two to four arguments. The preset `authentication_status` value is passed as the first parameter and may be updated by the stored procedure. The returned value of the first parameter is the `authentication_status` from the `authenticate_user` script.

4. If an `authenticate_user_hashed` script exists, then the following occurs:
 - a. If a password was provided, a hashed value is calculated for it. If a new password was provided, a hashed value is calculated for it.
 - b. The `authenticate_user_hashed` script is called with the current value of `authentication_status` (either the preset `authentication_status` if the `authenticate_user` script doesn't exist, or the `authentication_status` returned from the `authenticate_user` script) and the hashed passwords. The behavior is identical to step 3. The returned value of the first parameter is used as the `authentication_status` of the `authenticate_user_hashed` script.
5. The MobiLink server takes the greater value of the `auth_user` status returned from the `authenticate_user` script and `authenticate_user_hashed` script, if they exist, or the preset `authentication_status` if neither of the scripts exist.
6. The MobiLink server queries the `ml_user` table for the MobiLink user name you provided.
 - a. If either of the custom scripts `authenticate_user` or `authenticate_user_hashed` was called but the MobiLink user name you provided is not in the `ml_user` table and the `authentication_status` is valid (1000 or 2000), the MobiLink user name is added to the MobiLink system table `ml_user`. If `authentication_status` is not valid, `ml_user` is not updated and an error occurs.
 - b. If the custom scripts were not called and the MobiLink user name you provided is not in the `ml_user` table, the MobiLink user name you provided is added to `ml_user` if you started the MobiLink server with the `-zu+` option. Otherwise, an error occurs and `authentication_status` is set to be invalid.
 - c. If the custom scripts were called and the MobiLink user name you provided is in the `ml_user` table, nothing happens.
 - d. If the custom scripts were **not** called and the MobiLink user name you provided is in the `ml_user` table, the password is checked against the value in the `ml_user` table. If the password matches the one in the `ml_user` table for the MobiLink user, the `authentication_status` is set to be valid. Otherwise the `authentication_status` is set to be invalid.
7. If that `authentication_status` is valid and neither of the scripts `authenticate_user` or `authenticate_user_hashed` was called and you provided a new password in the `ml_user` table for this MobiLink user, the password is changed to the one you provided.

8. If you have defined an `authenticate_parameters` script and the `authentication_status` is valid (1000 or 2000), then the following occurs:
 - a. The parameters are passed to the `authenticate_parameters` script.
 - b. If the `authenticate_parameters` script returns an `authentication_status` value greater than the current `authentication_status`, the new `authentication_status` overwrites the old value.
9. If `authentication_status` is not valid, the synchronization is aborted.
10. If you have defined the `modify_user` script, it is called to replace the MobiLink user name you provided with a new MobiLink user name returned by this script.
11. The MobiLink server always commits the transaction after MobiLink user authentication, regardless of the `authentication_status`. If the `authentication_status` is valid (1000 or 2000), synchronization continues. If the `authentication_status` is invalid, the synchronization is aborted.

Custom user authentication

You can choose to use a user authentication mechanism other than the built-in MobiLink mechanism. The following are some reasons for using a custom user authentication mechanism:

- To include integration with existing database user authentication schemes or external authentication mechanisms.
- To supply custom features, such as minimum password length or password expiry, that do not exist in the built-in MobiLink mechanism.

There are three custom authentication tools:

- `mlsrv12 -zu+` option
- `authenticate_user` script or `authenticate_user_hashed` script
- `authenticate_parameters` script

The `mlsrv12 -zu+` option allows you to control the automatic addition of users. For example, specify `-zu+` to have all unrecognized MobiLink user names added to the `ml_user` table when they first synchronize. The `-zu+` option is only needed for built-in MobiLink authentication.

The `authenticate_user`, `authenticate_user_hashed`, and `authenticate_parameters` scripts override the default MobiLink user authentication mechanism. Any user who successfully authenticates is automatically added to the `ml_user` table.

You can use the `authenticate_user` script to create custom authentication of user IDs and passwords. If this script exists, it is executed instead of the built-in password comparison. The script must return error codes to indicate the success or failure of the authentication.

There are several predefined scripts for the `authenticate_user` event that are installed with MobiLink. These make it easier for you to authenticate using LDAP, POP3, and IMAP servers. See [“Authenticating to external servers” on page 15](#).

Use `authenticate_parameters` to create custom authentication that depends on values other than user IDs and passwords.

See also

- [“-zu mlsrv12 option” \[MobiLink - Server Administration\]](#)
- [“authenticate_user connection event” \[MobiLink - Server Administration\]](#)
- [“authenticate_user_hashed connection event” \[MobiLink - Server Administration\]](#)
- [“authenticate_parameters connection event” \[MobiLink - Server Administration\]](#)

Java and .NET user authentication

User authentication is a natural use of Java and .NET synchronization logic because Java and .NET classes allow you to reach out to other sources of user names and passwords used in your computing environment, such as application servers.

A simple sample is included in the directory `samples-dir\MobiLink\JavaAuthentication`. The sample code in `samples-dir\MobiLink\JavaAuthentication\CustEmpScripts.java` implements a simple user authentication system. On the first synchronization, a MobiLink user name is added to the `login_added` table. On subsequent synchronizations, a row is added to the `login_audit` table. In this sample, there is no test before adding a user ID to the `login_added` table. (For information about `samples-dir`, see [“Samples directory” \[SQL Anywhere Server - Database Administration\]](#).)

For a .NET sample that explains user authentication, see [“.NET synchronization example” \[MobiLink - Server Administration\]](#).

Authenticating to external servers

Predefined Java synchronization scripts are included with MobiLink that make it simpler for you to authenticate to external servers using the `authenticate_user` event. Predefined scripts are available for the following authentication servers:

- POP3 or IMAP servers using the JavaMail 1.2 API
- LDAP servers using the Java Naming and Directory Interface (JNDI)

How you use these scripts is determined by whether your MobiLink user names map directly to the user IDs in your external authentication system.

Note

You can also set up authentication to external servers in Sybase Central, using the Authentication tab. See [“MobiLink Plug-in for Sybase Central” \[MobiLink - Getting Started\]](#).

If your MobiLink user names map directly to your user IDs

In the simple case where the MobiLink user name maps directly to a valid user ID in your authentication system, the predefined scripts can be used directly in response to the `authenticate_user` connection event. The authentication code initializes itself based on properties stored in the `ml_property` table.

To use predefined scripts directly in `authenticate_user`

1. Add the predefined Java synchronization script to the `ml_scripts` MobiLink system table. You can do this using a stored procedure or in Sybase Central.

- To use the `ml_add_java_connection_script` stored procedure, run the following command:

```
call ml_add_java_connection_script(  
    'MyVersion',  
    'authenticate_user',  
    'iAnywhere.ml.authentication.ServerType.authenticate' )
```

where *MyVersion* is the name of a script version, and *ServerType* is **LDAP**, **POP3**, or **IMAP**.

- To use the **Add Connection Script Wizard** in Sybase Central, choose **authenticate_user** as the script type, and enter the following in the Code Editor:

```
iAnywhere.ml.authentication.ServerType.authenticate
```

where *ServerType* is **LDAP**, **POP3**, or **IMAP**.

See “[ml_add_java_connection_script system procedure](#)” [*MobiLink - Server Administration*].

2. Add properties for this authentication server.

Use the `ml_add_property` stored procedure for each property you need to set:

```
call ml_add_property(  
    'ScriptVersion',  
    'MyVersion',  
    'property_name',  
    'property_value' )
```

where *MyVersion* is the name of a script version, *property_name* is determined by your authentication server, and *property_value* is a value appropriate to your application. Repeat this call for every property you want to set.

See “[External authenticator properties](#)” on page 17 and “[ml_add_property system procedure](#)” [*MobiLink - Server Administration*].

If your MobiLink user names do not map directly to your user IDs

If your MobiLink user names are not equivalent to your user IDs, the code must be called indirectly and you must extract or map the user ID from the `ml_user` value. You do this by writing a Java class.

See “[Writing synchronization scripts in Java](#)” [*MobiLink - Server Administration*].

The following is a simple example. In this example, the code in the `extractUserID` method has been left out because it depends on how the `ml_user` value maps to a `userid`. All the work is done in the “`authenticate`” method of the authentication class.

```

package com.mycompany.mycode;

import anywhere.ml.authentication.*;
import anywhere.ml.script.*;

public class MLEvents
{
    private DBConnectionContext _context;
    private POP3 _pop3;

    public MLEvents( DBConnectionContext context )
    {
        _context = context;
        _pop3 = new POP3( context );
    }

    public void authenticateUser(
        InOutInteger status,
        String userID,
        String password,
        String newPassword )
    {
        String realUserID = extractUserID( userID );
        _pop3.authenticate( status, realUserID, password, newPassword );
    }

    private String extractUserID( String userID )
    {
        // code here to map ml_user to a "real" POP3 user
    }
}

```

In this example, The POP3 object needs to be initialized with the DBConnectContext object so that it can find its initialization properties. If you do not initialize it this way, you must set the properties in code. For example,

```

POP3 pop3 = new POP3();
pop3.setServerName( "smtp.sybase.com" );
pop3.setServerPort( 25 );

```

This applies to any of the authentication classes, although the properties vary by class.

External authenticator properties

MobiLink provides reasonable defaults wherever possible, especially in the LDAP case. The properties that can be set vary, but following are the basic ones.

POP3 authenticator

mail.pop3.host	the hostname of the server
mail.pop3.port	the port number (can be omitted if default 110 is used)

See <http://java.sun.com/products/javamail/javadocs/com/sun/mail/pop3/package-summary.html>.

IMAP authenticator

mail.imap.host	the hostname of the server
mail.imap.port	the port number (can be omitted if default 143 is used)

See <http://java.sun.com/products/javamail/javadocs/com/sun/mail/imap/package-summary.html>.

LDAP authenticator

java.naming.provider.url	the URL of the LDAP server, such as ldap://ops-yourLocation/dn=sybase,dn=com
--------------------------	------------------------------------------------------------------------------

For more information, see the JNDI documentation.

MobiLink client utilities

Introduction to MobiLink client utilities

There are two MobiLink client utilities:

- “Microsoft ActiveSync Provider Installation utility (mlasinst)” on page 18
- “MobiLink File Transfer utility (mlfiletransfer)” on page 21

In addition, see:

- UltraLite utilities: “UltraLite utilities” [*UltraLite - Database Management and Reference*]
- MobiLink server utilities: “MobiLink utilities” [*MobiLink - Server Administration*]
- Other SQL Anywhere utilities: “Database administration utilities” [*SQL Anywhere Server - Database Administration*]

Microsoft ActiveSync Provider Installation utility (mlasinst)

Installs a MobiLink provider for Microsoft ActiveSync (known as Windows Mobile Device Center on Windows Vista), or registers and installs UltraLite applications on Windows Mobile devices.

Syntax

`mlasinst [options] [[src] dst name class [args]]`

Options	Description
-d	Initially disable the application.

Options	Description
-k <i>path</i>	Specify the location of the desktop provider <i>mlasdesk.dll</i> . By default, the file is located in <i>install-dir\bin32</i> . End users (who generally do not have the full SQL Anywhere install) may need to specify -k when installing the MobiLink Microsoft ActiveSync provider.
-l <i>filename</i>	Specify the name of the activity log file.
-n	Register the application but do not copy it to the device. In addition to installing the MobiLink Microsoft ActiveSync provider, this option registers an application but does not copy it to the device. This is appropriate if the application includes more than one file (for example, if it is compiled to use the UltraLite runtime library DLL rather than a static library) or if you have an alternative method of copying the application to the device.
-t <i>n</i>	Specify how long, in seconds, the desktop provider should wait for a response from the client before timing out; the default is 30.
-u	Uninstall the MobiLink provider for Microsoft ActiveSync. This option unregisters all applications that have been registered for use with the MobiLink Microsoft ActiveSync provider and uninstalls the MobiLink Microsoft ActiveSync provider. No files are deleted from the desktop computer or the device by this operation. If the device is not connected to the desktop, an error is reported.
-v <i>path</i>	Specify the location of the device provider <i>mlasdev.dll</i> . By default, the file is looked for in a platform-specific directory in <i>install-dir\CE</i> . End users (who generally do not have the full SQL Anywhere install) may need to specify -v when installing the MobiLink Microsoft ActiveSync provider.

Other parameters	Description
<i>src</i>	Specify the source file name and path for copying an application to the device. Supply this parameter only if you are registering an application and copying it to the device. Do not supply the parameter if you use the -n option.
<i>dst</i>	Specify the destination file name and path on the device for an application.
<i>name</i>	Specify the name by which Microsoft ActiveSync refers to the application.
<i>class</i>	Specify the registered Windows class name of the application.

Other parameters	Description
<i>args</i>	Specify command line arguments to pass to the application when Microsoft ActiveSync starts the application.

Remarks

This utility installs a MobiLink provider for Microsoft ActiveSync. The provider includes both a component that runs on the desktop (*mlasdesk.dll*) and a component that is deployed to the Windows Mobile device (*mlasdev.dll*). The *mlasinst* utility makes a registry entry pointing to the current location of the desktop provider; and copies the device provider to the device.

If additional arguments are supplied, the *mlasinst* utility can also be used to register and install UltraLite applications onto a Windows Mobile device. Alternatively, you can register and install UltraLite applications using the Microsoft ActiveSync software.

Subject to licensing requirements, you may supply this application together with the desktop and device components to end users so that they can prepare their copies of your application for use with Microsoft ActiveSync.

You must be connected to a remote device to install the Microsoft ActiveSync provider.

For complete instructions on using the Microsoft ActiveSync Provider Installation utility, see:

- SQL Anywhere: [“Installing the MobiLink provider for Microsoft ActiveSync” on page 84](#)
- UltraLite: [“Using ActiveSync with UltraLite on Windows Mobile” \[UltraLite - Database Management and Reference\]](#)

Examples

The following command installs the MobiLink provider for Microsoft ActiveSync using default arguments. It does not register an application. The device must be connected to your desktop for the installation to succeed.

```
mlasinst
```

The following command uninstalls the MobiLink provider for Microsoft ActiveSync. The device must be connected to your desktop for the uninstall to succeed:

```
mlasinst -u
```

The following command installs the MobiLink provider for Microsoft ActiveSync, if it is not already installed, and registers the application *myapp.exe*. It also copies the *c:\My Files\myapp.exe* file to *\Program Files\myapp.exe* on the device. The *-p -x* arguments are command line options for *myapp.exe* when started by Microsoft ActiveSync. The command must be entered on a single line:

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

See also

- “Using Microsoft ActiveSync synchronization” on page 83
- “UltraLite synchronization parameters and network protocol options” [*UltraLite - Database Management and Reference*]

MobiLink File Transfer utility (mlfiletransfer)

Uploads or downloads a file through MobiLink.

Syntax

mlfiletransfer [*options*] *file*

Option	Description
-ap <i>param1</i> , ...	MobiLink authentication parameters. See “Authentication parameters” [<i>MobiLink - Server Administration</i>].
-g	Shows transfer progress.
-i	Ignore partial transfer from a previous attempt.
-k	Remote key to identify the remote. This is optional.
-lf <i>filename</i>	The local name of the file to be transferred. By default, the name as recognized by the server (i.e. <i>file</i>) is used.
-lp <i>path</i>	The local path for the file to be transferred. By default, the local path is the root directory on Windows Mobile, and the current directory on other platforms.
-p <i>password</i>	The password for the MobiLink user name.
-q	Quiet mode. Messages are not displayed.
-s	Upload a file to MobiLink. Download is the default.
-u <i>username</i>	MobiLink user name. This option is required.
-v <i>version</i>	The script version. This option is required.
-x <i>protocol (options)</i>	<p>The <i>protocol</i> can be one of tcpip, tls, http, or https. This option is required.</p> <p>The protocol-options you can use depend on the protocol. For a list of options for each protocol, see “MobiLink client network protocol option summary” on page 23.</p>

Option	Description
<i>file</i>	<p>The file to be transferred as named on the server. When downloading, MobiLink looks for the file in the <i>username</i> subdirectory of the -ftr directory; if it does not find it there, it looks in the -ftr directory. If the file is not in either place, MobiLink generates an error. See “-ftr mlsrv12 option” [MobiLink - Server Administration].</p> <p>When uploading, MobiLink looks for the files in the directory specified with the -ftru mlsrv12 option. See “-ftru mlsrv12 option” [MobiLink - Server Administration].</p>

Remarks

This utility is useful for downloading files when you first create a remote database, when you need to upgrade software on your remote device, and so on.

To use this utility to download files, you must start the MobiLink server with the -ftr option. The -ftr option creates a root directory for the file to be transferred, and creates a subdirectory for every registered MobiLink user.

To use this utility to upload files, you must start the MobiLink server with the -ftru option. The -ftru option creates a location for files that are to be uploaded.

You can use mlagent as an alternative to mlfiletransfer. See “[Configuring and running the MobiLink Agent on the client device](#)” [[MobiLink - Server Administration](#)].

UltraLite users can also use the MLFileDownload and MLFileUpload methods in the UltraLite runtime. See “[Using MobiLink file transfers](#)” [[UltraLite - Database Management and Reference](#)].

See also

- “-ftr mlsrv12 option” [[MobiLink - Server Administration](#)]
- “authenticate_file_transfer connection event” [[MobiLink - Server Administration](#)]

Example

The following command connects the MobiLink server to the CustDB sample database. The -ftr %SystemRoot%\system32 option tells the MobiLink server to start monitoring the *Windows\system32* directory for requested files. In this example the MobiLink server first looks for the file in the *C:\Windows\system32\mobilink-username* directory. If the file does not exist, it looks in the *C:\Windows\system32* directory. In general you would not want to have the MobiLink server monitor your *Windows\system32* folder for files. This example uses the *Windows\system32* directory so that it can transfer the Notepad utility, which is located there.

```
mksrv12 -c "dsn=SQL Anywhere 12 CustDB" -zu+ -ftr %SystemRoot%\system32
```

The following command runs the mlfiletransfer utility. It causes the MobiLink server to download *notepad.exe* to your local directory.


```
MLFileTransfer -u 1 -v "custdb 12.0" -x tcpip notepad.exe
```

MobiLink client network protocol options

MobiLink client network protocol option summary

This section describes the network protocol options you can use when connecting a MobiLink client to the MobiLink server. Several MobiLink client utilities use the MobiLink client network protocol options:

To use client network protocol options with...	See...
dbmlsync	“CommunicationAddress (adr) extended option” on page 128
UltraLite	“Stream Parameters synchronization parameter” [<i>UltraLite - Database Management and Reference</i>] or -x option in “UltraLite Synchronization utility (ulsync)” [<i>UltraLite - Database Management and Reference</i>]
UltraLiteJ	“Network protocol options for UltraLiteJ synchronization streams” [<i>UltraLiteJ</i>]
Relay Server	“Relay Server configuration file” [<i>Relay Server</i>]
MobiLink Monitor	“Starting the MobiLink Monitor” [<i>MobiLink - Server Administration</i>]
MobiLink file transfer	“MobiLink File Transfer utility (mlfiletransfer)” on page 21
MobiLink Listener	-x in “MobiLink Listener utility for Windows devices (dbsln)” [<i>MobiLink - Server-Initiated Synchronization</i>]
QAnywhere Agent	“-x qaagent option” [<i>QAnywhere</i>]

The network protocol you choose must match the synchronization protocol used by the client. For information about how to set connection options for the MobiLink server, see “-x mlsrv12 option” [*MobiLink - Server Administration*].

Protocol options

- **TCP/IP protocol options** If you specify the **tcpip** option, you can optionally specify the following protocol options:

TCP/IP protocol option	For more information, see...
<code>buffer_size=bytes</code>	“buffer_size” on page 28
<code>client_port=nnnnn[-mmmmm]</code>	“client_port” on page 32
<code>e2ee_type={rsa ecc}</code>	“e2ee_type” on page 35
<code>e2ee_public_key=file</code>	“e2ee_public_key” on page 36
<code>compression={zlib none}</code>	“compression” on page 33
<code>host=hostname</code>	“host” on page 38
<code>network_leave_open={off on}</code>	“network_leave_open” on page 45
<code>network_name=name</code>	“network_name” on page 45
<code>port=portnumber</code>	“port” on page 47
<code>timeout=seconds</code>	“timeout” on page 50
<code>zlib_download_window_size=window-bits</code>	“zlib_download_window_size” on page 56
<code>zlib_upload_window_size=window-bits</code>	“zlib_upload_window_size” on page 57

- **TCP/IP protocol with security** If you specify the `tls` option, which is TCP/IP with TLS security, you can optionally specify the following protocol options:

TLS protocol option	For more information, see...
<code>buffer_size=bytes</code>	“buffer_size” on page 28
<code>certificate_company=company_name</code>	“certificate_company” on page 29
<code>certificate_name=name</code>	“certificate_name” on page 30
<code>certificate_unit=company_unit</code>	“certificate_unit” on page 32
<code>client_port=nnnnn[-mmmmm]</code>	“client_port” on page 32
<code>compression={zlib none}</code>	“compression” on page 33
<code>e2ee_type={rsa ecc}</code>	“e2ee_type” on page 35
<code>e2ee_public_key=file</code>	“e2ee_public_key” on page 36
<code>fips={y n}</code>	“fips” on page 36

TLS protocol option	For more information, see...
host = <i>hostname</i>	“host” on page 38
identity = <i>filename</i>	“identity” on page 42
identity_password = <i>password</i>	“identity_password” on page 44
network_leave_open ={ off on }	“network_leave_open” on page 45
network_name = <i>name</i>	“network_name” on page 45
port = <i>portnumber</i>	“port” on page 47
timeout = <i>seconds</i>	“timeout” on page 50
tls_type ={ rsa ecc }	“tls_type” on page 51
trusted_certificates = <i>filename</i>	“trusted_certificates” on page 53
zlib_download_window_size = <i>window-bits</i>	“zlib_download_window_size” on page 56
zlib_upload_window_size = <i>window-bits</i>	“zlib_upload_window_size” on page 57

- **HTTP protocol** If you specify the **http** option, you can optionally specify the following protocol options:

HTTP protocol option	For more information, see...
buffer_size = <i>number</i>	“buffer_size” on page 28
client_port = <i>nnnnn</i> [- <i>mmmmm</i>]	“client_port” on page 32
compression ={ zlib none }	“compression” on page 33
custom_header = <i>header</i>	“custom_header” on page 34
e2ee_type ={ rsa ecc }	“e2ee_type” on page 35
e2ee_public_key = <i>file</i>	“e2ee_public_key” on page 36
http_password = <i>password</i>	“http_password” on page 39
http_proxy_password = <i>password</i>	“http_proxy_password” on page 40
http_proxy_userid = <i>userid</i>	“http_proxy_userid” on page 41
http_userid = <i>userid</i>	“http_userid” on page 42

HTTP protocol option	For more information, see...
host = <i>hostname</i>	“host” on page 38
network_leave_open ={ off on }	“network_leave_open” on page 45
network_name = <i>name</i>	“network_name” on page 45
persistent ={ off on }	“persistent” on page 47
port = <i>portnumber</i>	“port” on page 47
proxy_host = <i>proxy-hostname-or-ip</i>	“proxy_host” on page 48
proxy_port = <i>proxy-portnumber</i>	“proxy_port” on page 49
set_cookie = <i>cookie-name = cookie-value</i>	“set_cookie” on page 49
timeout = <i>seconds</i>	“timeout” on page 50
url_suffix = <i>suffix</i>	“url_suffix” on page 54
version = <i>HTTP-version-number</i>	“version” on page 55
zlib_download_window_size = <i>window-bits</i>	“zlib_download_window_size” on page 56
zlib_upload_window_size = <i>window-bits</i>	“zlib_upload_window_size” on page 57

- **HTTPS protocol** If you specify the **https** option, which is HTTP with RSA encryption, you can optionally specify the following protocol options:

HTTPS protocol option	For more information, see...
buffer_size = <i>number</i>	“buffer_size” on page 28
certificate_company = <i>company_name</i>	“certificate_company” on page 29
certificate_name = <i>name</i>	“certificate_name” on page 30
certificate_unit = <i>company_unit</i>	“certificate_unit” on page 32
client_port = <i>nnnnn[-nnnnnn]</i>	“client_port” on page 32
compression ={ zlib none }	“compression” on page 33
custom_header = <i>header</i>	“custom_header” on page 34
e2ee_type ={ rsa ecc }	“e2ee_type” on page 35

HTTPS protocol option	For more information, see...
e2ee_public_key = <i>file</i>	“e2ee_public_key” on page 36
fips ={ <i>y n</i> }	“fips” on page 36
host = <i>hostname</i>	“host” on page 38
http_password = <i>password</i>	“http_password” on page 39
http_proxy_password = <i>password</i>	“http_proxy_password” on page 40
http_proxy_userid = <i>userid</i>	“http_proxy_userid” on page 41
http_userid = <i>userid</i>	“http_userid” on page 42
identity = <i>filename</i>	“identity” on page 42
identity_password = <i>password</i>	“identity_password” on page 44
network_leave_open ={ <i>off on</i> }	“network_leave_open” on page 45
network_name = <i>name</i>	“network_name” on page 45
persistent ={ <i>off on</i> }	“persistent” on page 47
port = <i>portnumber</i>	“port” on page 47
proxy_host = <i>proxy-hostname-or-ip</i>	“proxy_host” on page 48
proxy_port = <i>proxy-portnumber</i>	“proxy_port” on page 49
set_cookie = <i>cookie-name = cookie-value</i>	“set_cookie” on page 49
timeout = <i>seconds</i>	“timeout” on page 50
tls_type ={ <i>rsa ecc</i> }	“tls_type” on page 51
trusted_certificates = <i>filename</i>	“trusted_certificates” on page 53
url_suffix = <i>suffix</i>	“url_suffix” on page 54
version = <i>HTTP-version-number</i>	“version” on page 55
zlib_download_window_size = <i>window-size</i>	“zlib_download_window_size” on page 56
zlib_upload_window_size = <i>window-bits</i>	“zlib_upload_window_size” on page 57

Separately licensed component required

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

See “[Separately licensed components](#)” [*SQL Anywhere 12 - Introduction*].

buffer_size

Specify the maximum number of bytes to buffer before writing to the network. For HTTP and HTTPS, this translates to the maximum HTTP request body size.

Syntax

`buffer_size=bytes`

Available protocols

- TCPIP, TLS, HTTP, HTTPS

Default

- Windows Mobile - 16K
- All other platforms - 64K

Remarks

In general for HTTP and HTTPS, the larger the buffer size, the fewer the number of HTTP request-response cycles, but the more memory required.

For TCPIP and TLS, it is also the case that a larger size performs faster but requires more memory; however, the performance difference is less significant than for HTTP.

Units are in bytes. Specify K for kilobytes, M for megabytes or G for gigabytes.

The maximum value is 1G.

This option controls the size of the requests from the client and has no bearing on the size of the responses from MobiLink.

For information about how to set network protocol options with dbmlsync, see “[CommunicationAddress \(adr\) extended option](#)” on page 128.

For information about how to set network protocol options with UltraLite, see “[Network protocol options for UltraLite synchronization streams](#)” [*UltraLite - Database Management and Reference*].

Example

The following example sets the maximum number of bytes to 32K.

On a SQL Anywhere client, the implementation is:

```
dbmlsync -e "adr=buffer_size=32K"
```

In an UltraLite application written in embedded SQL or C++, the implementation is:

```
synch_info.stream_params = TEXT("buffer_size=32K");
```

certificate_company

If specified, the application only accepts server certificates when the Organization field on the certificate matches this value.

Separately licensed component required

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

See “[Separately licensed components](#)” [*SQL Anywhere 12 - Introduction*].

Syntax

```
certificate_company=organization
```

Available protocols

- TLS, HTTPS

Default

None

Remarks

MobiLink clients trust all certificates signed by the certificate authority, so they may also trust certificates that the same certificate authority has issued to other companies. Without a means to discriminate, your clients might mistake a competitor's MobiLink server for your own and accidentally send it sensitive information. This option specifies a further level of verification, that the Organization field in the identity portion of the certificate also matches a value you specify.

For information about how to set network protocol options with dbmlsync, see “[CommunicationAddress \(adr\) extended option](#)” on page 128.

For information about how to set network protocol options with UltraLite, see “[Network protocol options for UltraLite synchronization streams](#)” [*UltraLite - Database Management and Reference*].

See also

- “Encrypting MobiLink client/server communications” [[SQL Anywhere Server - Database Administration](#)]
- “Verifying certificate fields” [[SQL Anywhere Server - Database Administration](#)]
- “-x mlsrv12 option” [[MobiLink - Server Administration](#)]
- “trusted_certificates” on page 53
- “certificate_name” on page 30
- “certificate_unit” on page 32

Example

The following examples tell a SQL Anywhere client to check all three identity fields and to accept only the named values. This example verifies all three fields. You can instead choose to verify only one or two fields.

For example, if you have SQL Anywhere clients you can set up certificate verification in the subscription as follows:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
FOR user01
TO test_pub
ADDRESS 'port=3333;
trusted_certificates=certicom.crt;
certificate_company=Sybase, Inc.;
certificate_unit=iAnywhere;certificate_name=sample'
```

In an UltraLite application written in embedded SQL in C or C++, you can set up certificate verification as follows, assuming that the trusted certificate was installed in the database when the database was created:

```
ul_sync_info info;
info.stream = "tls";
info.stream_parms = UL_TEXT("port=9999;")
    UL_TEXT ( "certificate_company=Sybase, Inc.;" )
    UL_TEXT ( "certificate_unit=iAnywhere;" )
    UL_TEXT ( "certificate_name=sample;" );
...
ULSynchronize( &info );
```

certificate_name

If specified, the application only accepts server certificates when the Common Name field on the certificate matches this value.

Separately licensed component required

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

See “Separately licensed components” [[SQL Anywhere 12 - Introduction](#)].

Syntax

certificate_name=*common-name*

Available protocols

- TLS, HTTPS

Default

None

Remarks

For information about how to set network protocol options with dbmlsync, see [“CommunicationAddress \(adr\) extended option”](#) on page 128.

For information about how to set network protocol options with UltraLite, see [“Network protocol options for UltraLite synchronization streams”](#) [*UltraLite - Database Management and Reference*].

See also

- [“Encrypting MobiLink client/server communications”](#) [*SQL Anywhere Server - Database Administration*]
- [“Verifying certificate fields”](#) [*SQL Anywhere Server - Database Administration*]
- [“-x mlsrv12 option”](#) [*MobiLink - Server Administration*]
- [“trusted_certificates”](#) on page 53
- [“certificate_company”](#) on page 29
- [“certificate_unit”](#) on page 32

Example

The following example sets up RSA encryption for an HTTPS protocol. This requires setup on the server and client. Each command must be written on one line.

On the server, the implementation is:

```
mlsrv12
-c "dsn=SQL Anywhere 12 Demo;uid=DBA;pwd=sql"
-x https(
  port=9999;
  identity=c:\sa12\bin32\rsaserver.id;
  identity_password=test)
```

On a SQL Anywhere client, the implementation is:

```
dbmlsync
-c "dsn=mydb;uid=DBA;pwd=sql"
-e "ctp=https;
  adr='port=9999;
  trusted_certificates=c:\sa12\bin32\rsaroot.crt;
  certificate_name=RSA Server'"
```

On an UltraLite client, the implementation is:

```
info.stream = "https";
info.stream_parms = TEXT(
"port=9999;
trusted_certificates=\sa12\bin32\rsaroot.crt;
certificate_name=RSA Server");
```

certificate_unit

If specified, the application only accepts server certificates when the Organization Unit field on the certificate matches this value.

Separately licensed component required

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

See “Separately licensed components” [[SQL Anywhere 12 - Introduction](#)].

Syntax

`certificate_unit=organization-unit`

Available protocols

- TLS, HTTPS

Default

None

Remarks

For information about how to set network protocol options with dbmlsync, see “[CommunicationAddress \(adr\) extended option](#)” on page 128.

For information about how to set network protocol options with UltraLite, see “[Network protocol options for UltraLite synchronization streams](#)” [[UltraLite - Database Management and Reference](#)].

See also

- “[Encrypting MobiLink client/server communications](#)” [[SQL Anywhere Server - Database Administration](#)]
- “[Verifying certificate fields](#)” [[SQL Anywhere Server - Database Administration](#)]
- “[-x mlsrv12 option](#)” [[MobiLink - Server Administration](#)]
- “[trusted_certificates](#)” on page 53
- “[certificate_company](#)” on page 29
- “[certificate_name](#)” on page 30

Example

For examples of security, see “[certificate_name](#)” on page 30 and “[trusted_certificates](#)” on page 53.

client_port

Specify a range of client ports for communication.

Syntax

```
client_port=nnnn[-mmmmm]
```

Available protocols

- TCPIP, TLS, HTTP, HTTPS

Default

None

Remarks

Specify a low value and a high value to create a range of possible port numbers. To restrict the client to a specific port number, specify the same number for *nnnnn* and *mmmmm*. If you specify only one value, the end of the range is 100 greater than the initial value, for a total of 101 ports.

The option can be useful for clients inside a firewall communicating with a MobiLink server outside the firewall.

For information about how to set network protocol options with dbmlsync, see [“CommunicationAddress \(adr\) extended option” on page 128](#).

For information about how to set network protocol options with UltraLite, see [“Network protocol options for UltraLite synchronization streams” \[UltraLite - Database Management and Reference\]](#).

Example

The following example sets a 10000 port range of allowable client ports.

On a SQL Anywhere client, the implementation is:

```
dbmlsync -e "adr=client_port=10000-19999"
```

In an UltraLite application written in embedded SQL or C++, the implementation is:

```
synch_info.stream_parms = TEXT("client_port=10000-19999");
```

compression

Turns on or off compression of the synchronization stream between the MobiLink server and MobiLink clients.

Syntax

```
compression= { zlib | none }
```

Available protocols

- TCPIP, TLS, HTTP, HTTPS

Default

For UltraLite, compression is off by default.

For dbmsync, zlib compression is used by default.

Caution

In SQL Anywhere clients, if you turn off compression the data is completely unobfuscated; if security is an issue, you should encrypt the stream.

See “[Transport-layer security](#)” [*SQL Anywhere Server - Database Administration*].

Remarks

When you use zlib compression, you can configure the upload and download compression using the `zlib_download_window_size` option and `zlib_upload_window_size` option. Using these options, you can also turn off compression for either the upload or the download.

To use zlib compression in UltraLite, `mlczlib12.dll` must be deployed and for C++ only, applications must call `ULEnableZlibSyncCompression(sqlca)`.

See also

- “[zlib_download_window_size](#)” on page 56
- “[zlib_upload_window_size](#)” on page 57

Example

The following option sets compression for upload only, and sets the upload window size to 9:

```
"compression=zlib;zlib_download_window_size=0;zlib_upload_window_size=9"
```

custom_header

Specify a custom HTTP header.

Syntax

`custom_header=header`

HTTP headers are of the form *header-name: header-value*.

Available protocols

- HTTP, HTTPS

Default

None

Remarks

When you specify custom HTTP headers, the client includes the headers with every HTTP request it sends. To specify more than one custom header, use `custom_header` multiple times, using the semicolon (;) as a divider. For example: `custom_header=header1:value1; customer_header=header2:value2`

Custom headers are useful when your synchronization client interacts with a third-party tool that requires custom headers.

For information about how to set network protocol options with dbmlsync, see [“CommunicationAddress \(adr\) extended option” on page 128](#).

For information about how to set network protocol options with UltraLite, see [“Network protocol options for UltraLite synchronization streams” \[UltraLite - Database Management and Reference\]](#).

Example

Some HTTP proxies require all requests to contain special headers. The following example sets a custom HTTP header called MyProxyHdr to the value ProxyUser in an embedded SQL or C++ UltraLite application:

```
info.stream = "http";
info.stream_parms = TEXT(
    "host=www.myhost.com;proxy_host=www.myproxy.com;
    custom_header=MyProxyHdr:ProxyUser");
```

e2ee_type

Specify the asymmetric algorithm to use for key exchange for end-to-end encryption.

Syntax

```
e2ee_type= { rsa | ecc }
```

Available protocols

TCPIP, TLS, HTTP, HTTPS

Default

RSA

Remarks

Must be either **rsa** or **ecc** and must match the value specified on the server.

See also

- [“e2ee_public_key” on page 36](#)
- [“-x mlsrv12 option” \[MobiLink - Server Administration\]](#)
- [“Key Pair Generator utility \(createkey\)” \[SQL Anywhere Server - Database Administration\]](#)

Example

The following example shows end-to-end encryption for an UltraLite client:

```
info.stream = "https";
info.stream_parms =
    "tls_type=rsa;trusted_certificates=rsaroot.crt;e2ee_type=rsa;e2ee_public_key=
    rsapublic.pem"
```

e2ee_public_key

Specify the file containing the server's PEM-encoded public key for end-to-end encryption.

Syntax

`e2ee_public_key=file`

Available protocols

TCPIP, TLS, HTTP, HTTPS

Default

None

Remarks

The key type must match the type specified in the `e2ee_type` parameter.

This option is required for end-to-end encryption to take effect.

End-to-end encryption can also be used with the TLS/HTTPS protocol option `fips`. This option is not supported when using ECC. See [“fips” on page 36](#).

See also

- [“e2ee_type” on page 35](#)
- [“-x mlsrv12 option” \[MobiLink - Server Administration\]](#)
- [“Key Pair Generator utility \(createkey\)” \[SQL Anywhere Server - Database Administration\]](#)

Example

The following example shows end-to-end encryption for an UltraLite client:

```
info.stream = "https";
info.stream_parms =
"tls_type=rsa;trusted_certificates=rsaroot.crt;e2ee_type=rsa;e2ee_public_key=
rsapublic.pem"
```

fips

Use FIPS-approved encryption implementations for TLS encryption and end-to-end encryption.

Separately licensed component required

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

See [“Separately licensed components” \[SQL Anywhere 12 - Introduction\]](#).

Syntax

`fips={ y | n }`

Available protocols

HTTPS, TLS

Default

No

Remarks

FIPS is only supported for RSA encryption.

Non-FIPS clients can connect to FIPS servers and vice versa.

This option can be used with end-to-end encryption. If `fips` is set to `y`, MobiLink clients use FIPS 140-2 certified implementations of RSA and AES. This option is not supported when using ECC. See [“e2ee_type” on page 35](#) and [“e2ee_public_key” on page 36](#).

For information about how to set network protocol options with `dbmlsync`, see [“CommunicationAddress \(adr\) extended option” on page 128](#).

See also

- [“tls_type” on page 51](#)
- [“e2ee_type” on page 35](#)

Example

The following example sets up FIPS-approved RSA encryption for a TCP/IP protocol. This requires setup on the server and client. Each command must be written on one line.

On the server, the implementation is:

```
mlsrv12
-c "dsn=SQL Anywhere 12 Demo;uid=DBA;pwd=sql"
-x tls(
  port=9999;
  tls_type=rsa;
  fips=y;
  identity=c:\sa12\bin32\rsaserver.id;
  identity_password=test)
```

On a SQL Anywhere client, the implementation is:

```
dbmlsync -e
"CommunicationType=tls;
CommunicationAddress=
'tls_type=rsa;
fips=y;
trusted_certificates=\rsaroot.crt;
certificate_name=RSA Server'"
```

In an UltraLite application written in embedded SQL in C or C++, the implementation is:

```
info.stream = "tls";
info.stream_parms = TEXT(
"tls_type=rsa;
fips=y;
```

```
trusted_certificates=\rsaroot.crt;  
certificate_name=RSA Server");
```

host

Specify the host name or IP number for the computer on which the MobiLink server is running, or, if you are synchronizing through a web server, the computer where the web server is running.

Syntax

host=hostname-or-ip

Available protocols

- TCPIP, TLS, HTTP, HTTPS

Default

- Windows Mobile - the default value is the IP address of the desktop computer the device has an Microsoft ActiveSync partnership with.
- All other devices - the default is **localhost**.

Remarks

On Windows Mobile, do not use localhost, which refers to the remote device itself. The default value allows a Windows Mobile device to connect to a MobiLink server on the desktop computer to which the Windows Mobile device has an Microsoft ActiveSync partnership.

For information about how to set network protocol options with dbmlsync, see [“CommunicationAddress \(adr\) extended option” on page 128](#).

For information about how to set network protocol options with UltraLite, see [“Network protocol options for UltraLite synchronization streams” \[UltraLite - Database Management and Reference\]](#).

Example

In the following example, the client connects to a computer called myhost at port 1234.

On a SQL Anywhere client, the implementation is:

```
dbmlsync -e "adr='host=myhost;port=1234'"
```

In an UltraLite application written in embedded SQL or C++, the implementation is:

```
synch_info.stream_parms = TEXT("host=myhost;port=1234");
```

http_buffer_responses

When on, this option streams HTTP packets from MobiLink into a buffer before processing them, instead of processing the bytes immediately as they are received.

Syntax

`http_buffer_responses={ off | on }`

Available protocols

- HTTP, HTTPS

Default

Off

Remarks

Because of the extra memory overhead required, this feature should only be used to work around HTTP synchronization stability issues. In particular, the Microsoft ActiveSync proxy server for Windows Mobile devices throws away data that is not read within 15 seconds after the server has closed its side of the connection. Because MobiLink clients process the download as they receive it from MobiLink, there is a chance they will fail to finish reading an HTTP packet within the allotted 15 seconds, causing synchronization to fail with a stream error code when synchronizing using non-persistent HTTP. By specifying `http_buffer_responses=on` the client reads each HTTP packet in its entirety into a buffer before processing any of it, thereby working around the 15 second timeout.

http_password

Authenticate to third-party HTTP servers and gateways using RFC 2617 Basic or Digest authentication.

Syntax

`http_password=password`

Available protocols

- HTTP, HTTPS

Default

None

Remarks

This feature supports Basic and Digest authentication as described in RFC 2617.

With Basic authentication, passwords are included in HTTP headers in clear text; however, you can use HTTPS to encrypt the headers and protect this password. With Digest authentication, headers are not sent in clear text but are hashed.

You must use `http_userid` with this option.

For information about how to set network protocol options with dbmsync, see [“CommunicationAddress \(adr\) extended option” on page 128](#).

For information about how to set network protocol options with UltraLite, see [“Network protocol options for UltraLite synchronization streams” \[UltraLite - Database Management and Reference\]](#).

See also

- [“http_userid” on page 42](#)
- [“http_proxy_password” on page 40](#)
- [“http_proxy_userid” on page 41](#)

Example

The following example of an embedded SQL or C++ UltraLite application provides a user ID and password for basic authentication to a web server.

```
synch_info.stream = "https";  
synch_info.stream_parms = TEXT("http_userid=user;http_password=pwd");
```

http_proxy_password

Authenticate to third-party HTTP proxies using RFC 2617 Basic or Digest authentication.

Syntax

`http_proxy_password=password`

Available protocols

- HTTP, HTTPS

Default

None

Remarks

This feature supports Basic and Digest authentication as described in RFC 2617.

With Basic authentication, passwords are included in HTTP headers in clear text; you can use HTTPS, but the initial connection to the proxy is through HTTP, so this password is clear text. With Digest authentication, headers are not sent in clear text but are hashed.

You must use `http_proxy_userid` with this option.

For information about how to set network protocol options with dbmlsync, see [“CommunicationAddress \(adr\) extended option” on page 128](#).

For information about how to set network protocol options with UltraLite, see [“Network protocol options for UltraLite synchronization streams” \[UltraLite - Database Management and Reference\]](#).

See also

- [“http_password” on page 39](#)
- [“http_userid” on page 42](#)
- [“http_proxy_userid” on page 41](#)

Example

The following example of an embedded SQL or C++ UltraLite application provides a user ID and password for basic authentication to a web proxy.

```
synch_info.stream = "https";  
synch_info.stream_parms =  
TEXT("http_proxy_userid=user;http_proxy_password=pwd");
```

http_proxy_userid

Authenticate to third-party HTTP proxies using RFC 2617 Basic or Digest authentication.

Syntax

http_proxy_userid=*userid*

Available protocols

- HTTP, HTTPS

Default

None

Remarks

This feature supports Basic and Digest authentication as described in RFC 2617.

With Basic authentication, passwords are included in HTTP headers in clear text; you can use HTTPS, but the initial connection to the proxy is through HTTP, so the password is clear text. With Digest authentication, headers are not sent in clear text but are hashed.

You must use `http_proxy_password` with this option.

For information about how to set network protocol options with `dbmlsync`, see [“CommunicationAddress \(adr\) extended option” on page 128](#).

For information about how to set network protocol options with UltraLite, see [“Network protocol options for UltraLite synchronization streams” \[UltraLite - Database Management and Reference\]](#).

See also

- [“http_password” on page 39](#)
- [“http_userid” on page 42](#)
- [“http_proxy_password” on page 40](#)

Example

The following example of an embedded SQL or C++ UltraLite application provides a user ID and password for basic authentication to a web proxy.

```
synch_info.stream = "https";  
synch_info.stream_parms =  
TEXT("http_proxy_userid=user;http_proxy_password=pwd");
```

http_userid

Authenticate to third-party HTTP servers and gateways using RFC 2617 Basic or Digest authentication.

Syntax

`http_userid=userid`

Available protocols

- HTTP, HTTPS

Default

None

Remarks

This feature supports Basic and Digest authentication as described in RFC 2617.

With Basic authentication, passwords are included in HTTP headers in clear text; however, you can use HTTPS to encrypt the headers and protect the password. With Digest authentication, headers are not sent in clear text but are hashed.

You must use `http_password` with this option.

For information about how to set network protocol options with `dbmlsync`, see [“CommunicationAddress \(adr\) extended option” on page 128](#).

For information about how to set network protocol options with UltraLite, see [“Network protocol options for UltraLite synchronization streams” \[UltraLite - Database Management and Reference\]](#).

See also

- [“http_password” on page 39](#)
- [“http_proxy_password” on page 40](#)
- [“http_proxy_userid” on page 41](#)

Example

The following example of an embedded SQL or C++ UltraLite application provides a user ID and password for basic authentication to a web server.

```
synch_info.stream = "https";  
synch_info.stream_parms = TEXT("http_userid=user;http_password=pwd");
```

identity

Use this option to enable the use of client-side certificates to authenticate MobiLink clients to third party servers and proxies.

Syntax

identity=*filename*

Available protocols

RSA TLS, HTTPS

Default

None

Remarks

The *filename* indicates the file that contains the client's identity. An identity consists of the client certificate, the corresponding private key, and, optionally, the certificates of the intermediary certificate authorities.

If the private key is encrypted, use the `identity_password` option to specify a password. See [“identity_password” on page 44](#).

MobiLink clients cannot authenticate directly to MobiLink using client-side certificates. They can only be used to authenticate to third-party servers and proxies that have been configured to accept client-side certificate authentication, and are sitting between the client and MobiLink server.

This option is not available with ECC TLS.

This option is not supported in UltraLiteJ.

identity_name

This feature supports authentication using client identities (a certificate plus a private key) from Common Access Cards (CACs). This feature is only supported for Windows Mobile.

This parameter is used to specify the common name of the public certificate.

Separately licensed component required

This feature is part of the CAC Authentication Add-on and requires a separate license. See [“Separately licensed components” \[SQL Anywhere 12 - Introduction\]](#).

Syntax

identity_name=*name*

Available protocols

- TLS, HTTPS

Default

None.

Remarks

This parameter can only be used with FIPS-approved RSA encryption.

The public certificate must be installed in the device's certificate store.

Separately licensed component required

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

See “Separately licensed components” [[SQL Anywhere 12 - Introduction](#)].

identity_password

The password used to encrypt the private key found in the identity file.

Syntax

`identity_password=password`

Available protocols

- RSA TLS, HTTPS

Default

None.

Remarks

This option is only required if the private key in the identity file is encrypted. See “[identity](#)” on page 42.

This option is not available with ECC TLS.

network_adapter_name

Allows MobiLink clients (with the exception of UltraLiteJ) to explicitly specify the name of the network adapter to use to connect to MobiLink.

Syntax

`network_adapter_name=name`

Available protocols

- TCPIP, TLS, HTTP, HTTPS

Default

None

Remarks

This option is valid only for Windows Mobile and Windows desktop.

network_leave_open

When you specify `network_name`, you can optionally specify that the network connectivity should be left open after the synchronization finishes.

Syntax

```
network_leave_open={ off | on }
```

Available protocols

- TCPIP, TLS, HTTP, HTTPS

Default

The default is **off**.

Remarks

You must specify `network_name` to use this option.

When this option is set to on, network connectivity is left open after the synchronization finishes.

For information about how to set network protocol options with `dbmlsync`, see [“CommunicationAddress \(adr\) extended option” on page 128](#).

For information about how to set network protocol options with UltraLite, see [“Network protocol options for UltraLite synchronization streams” \[UltraLite - Database Management and Reference\]](#).

See also

- [“network_name” on page 45](#)

Example

In the following example, the client uses the network name `MyNetwork` and specifies that the connection should be left open after the synchronization finishes.

On a SQL Anywhere client, the implementation is:

```
dbmlsync -e "adr='network_name=MyNetwork;network_leave_open=on' "
```

In an UltraLite application written in embedded SQL or C++, the implementation is:

```
synch_info.stream_parms =  
TEXT("network_name=MyNetwork;network_leave_open=on");
```

network_name

Specify the network name to start if an attempt to connect to the network fails.

Syntax

network_name=*name*

Available protocols

- TCPIP, TLS, HTTP, HTTPS

Default

None

Remarks

Specify the network name so that you can use the MobiLink auto-dial feature. This allows you to connect from a Windows Mobile device or Windows desktop computer without manually dialing. Auto-dial is a secondary attempt to connect to the MobiLink server; first, the client attempts to connect without dialing, and if that fails and a *network_name* is specified, auto-dial is activated. When used with scheduling, your remote can synchronize unattended. When used without scheduling, this allows you to run `dbmlsync` without manually dialing a connection.

On Windows Mobile, the *name* should be one of the network profiles from the dropdown list in Settings » Connections » Connections. To use whatever you have set as your default for the internet network or work network, set the name to the keyword **default_internet** or **default_work**, respectively.

On Windows desktop platforms, the *name* should be one of the network profiles from Network & Dialup Connections.

For information about how to set network protocol options with `dbmlsync`, see [“CommunicationAddress \(adr\) extended option”](#) on page 128.

For information about how to set network protocol options with UltraLite, see [“Network protocol options for UltraLite synchronization streams”](#) [*UltraLite - Database Management and Reference*].

See also

- [“Scheduling synchronization”](#) on page 87
- [“network_leave_open”](#) on page 45

Example

In the following example, the client uses the network name `MyNetwork` and specifies that the connection should be left open after the synchronization finishes.

On a SQL Anywhere client, the implementation is:

```
dbmlsync -e "adr='network_name=MyNetwork;network_leave_open=on' "
```

In an UltraLite application written in embedded SQL or C++, the implementation is:

```
synch_info.stream_parms =  
TEXT( "network_name=MyNetwork;network_leave_open=on" );
```


persistent

Use a single TCP/IP connection for all HTTP requests in a synchronization.

Syntax

persistent={ off | on }

Available protocols

- HTTP, HTTPS

Default

Off

Remarks

The On value means that the client attempts to use the same TCP/IP connection for all HTTP requests in a synchronization. A setting of off is usually more compatible with intermediate agents.

You should only set persistent to on if you are connecting directly to MobiLink. If you are connecting through an intermediate agent such as a proxy, a persistent connection may cause problems.

For information about how to set network protocol options with dbmlsync, see [“CommunicationAddress \(adr\) extended option” on page 128](#).

For information about how to set network protocol options with UltraLite, see [“Network protocol options for UltraLite synchronization streams” \[UltraLite - Database Management and Reference\]](#).

port

Specify the socket port number of the MobiLink server.

Syntax

port=*port-number*

Available protocols

- TCP/IP, TLS, HTTP, HTTPS

Default

For TCP/IP, the default is **2439**, which is the IANA-registered port number for the MobiLink server.

For HTTP, the default is **80**.

For HTTPS, the default is **443**.

Remarks

The port number must be a decimal number that matches the port the MobiLink server is set up to listen on.

If you are synchronizing through a web server, specify the web server port accepting HTTP or HTTPS requests.

For information about how to set network protocol options with dbmlsync, see [“CommunicationAddress \(adr\) extended option” on page 128](#).

For information about how to set network protocol options with UltraLite, see [“Network protocol options for UltraLite synchronization streams” \[UltraLite - Database Management and Reference\]](#).

Example

In the following example, the client connects to a computer called myhost at port 1234.

On a SQL Anywhere client, the implementation is:

```
dbmlsync -e "adr='host=myhost;port=1234'"
```

In an UltraLite application written in embedded SQL or C++, the implementation is:

```
synch_info.stream_parms = TEXT("host=myhost;port=1234");
```

proxy_host

Specify the host name or IP address of the proxy server.

Syntax

proxy_host=*proxy-hostname-or-ip*

Available protocols

- HTTP, HTTPS

Default

None

Remarks

Use only if going through an HTTP proxy.

For information about how to set network protocol options with dbmlsync, see [“CommunicationAddress \(adr\) extended option” on page 128](#).

For information about how to set network protocol options with UltraLite, see [“Network protocol options for UltraLite synchronization streams” \[UltraLite - Database Management and Reference\]](#).

Example

In the following example, the client connects to a proxy server running on a computer called myproxyhost at port 1234.

On a SQL Anywhere Client, the implementation is:

```
dbmlsync -e "adr='proxy_host=myproxyhost;proxy_port=1234'"
```

In an UltraLite application written in embedded SQL or C++, the implementation is:

```
synch_info.stream_parms = TEXT("proxy_host=myproxyhost;proxy_port=1234");
```

proxy_port

Specify the port number of the proxy server.

Syntax

proxy_port=*proxy-port-number*

Available protocols

- HTTP, HTTPS

Default

None

Remarks

Use only if going through an HTTP proxy.

For information about how to set network protocol options with dbmlsync, see [“CommunicationAddress \(adr\) extended option” on page 128](#).

For information about how to set network protocol options with UltraLite, see [“Network protocol options for UltraLite synchronization streams” \[UltraLite - Database Management and Reference\]](#).

Example

In the following example, the client connects to a proxy server running on a computer called myproxyhost at port 1234.

On a SQL Anywhere Client, the implementation is:

```
dbmlsync -e "adr='proxy_host=myproxyhost;proxy_port=1234'"
```

In an UltraLite application written in embedded SQL or C++, the implementation is:

```
synch_info.stream_parms = TEXT("proxy_host=myproxyhost;proxy_port=1234");
```

set_cookie

Specify custom HTTP cookies to set in the HTTP requests used during synchronization.

Syntax

set_cookie=*cookie-name=cookie-value,...*

Available protocols

- HTTP, HTTPS

Default

None

Remarks

Custom HTTP cookies are useful when your synchronization client interacts with a third-party tool, such as an authentication tool, that uses cookies to identify sessions. For example, you have a system where a user agent connects to a web server, proxy, or gateway and authenticates itself. If successful, the agent receives one or more cookies from the server. The agent then starts a synchronization and hands over its session cookies through the `set_cookie` option.

If you have multiple name-value pairs, separate them with commas.

For information about how to set network protocol options with `dbmsync`, see [“CommunicationAddress \(adr\) extended option” on page 128](#).

For information about how to set network protocol options with UltraLite, see [“Network protocol options for UltraLite synchronization streams” \[UltraLite - Database Management and Reference\]](#).

Example

The following example sets a custom HTTP cookie in an embedded SQL or C++ UltraLite application.

```
info.stream = "http";
info.stream_parms = TEXT(
  "host=www.myhost.com;
  set_cookie=MySessionID=12345, enabled=yes;");
```

timeout

Specify the amount of time, in seconds, that the client waits for network operations to succeed before giving up.

Syntax

`timeout=seconds`

Available protocols

- TCP/IP, TLS, HTTP, HTTPS

Default

240 seconds

Remarks

If any connect, read, or write attempt fails to complete within the specified time, the client fails the synchronization.

Throughout the synchronization, the client sends liveness updates within the specified interval to let the MobiLink server know that it is still alive, and MobiLink sends back liveness updates to let the client know that it is still alive. To prevent slow networks from delaying the timeout past the specified time, MobiLink clients send keep-alive bytes to the MobiLink server at an interval of half the timeout value. When this value is set to 240 seconds, the keep-alive message is sent every 120 seconds.

You should be careful about setting the timeout to too low a value. Liveness checking increases network traffic because the MobiLink server and the client must communicate within each timeout period to ensure that the connection is still active. If the network or server load is very heavy and the timeout period is very short, a live connection could be abandoned because the MobiLink server and dbmlsync were unable to confirm that the connection is still active. The liveness timeout should generally not be less than 30 seconds.

The maximum timeout is 10 minutes. You can specify a larger number than 600 seconds, but it is interpreted as 600 seconds.

The value 0 means that the timeout is 10 minutes.

For information about how to set network protocol options with dbmlsync, see [“CommunicationAddress \(adr\) extended option”](#) on page 128.

For information about how to set network protocol options with UltraLite, see [“Network protocol options for UltraLite synchronization streams”](#) [*UltraLite - Database Management and Reference*].

Example

The following example sets the timeout to 300 seconds.

On a SQL Anywhere client, the implementation is:

```
dbmlsync -e "adr=timeout=300"
```

In an UltraLite application written in embedded SQL or C++, the implementation is:

```
synch_info.stream_parms = TEXT("timeout=300");
```

tls_type

Specify the encryption cipher to use for synchronization.

Separately licensed component required

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

See [“Separately licensed components”](#) [*SQL Anywhere 12 - Introduction*].

Syntax

```
tls_type=cipher
```

Available protocols

- TLS, HTTPS

Default

RSA

Remarks

All communication for this synchronization is to be encrypted using the specified cipher. The cipher can be one of:

- **ecc** for elliptic-curve encryption.
- **rsa** for RSA encryption.

For information about how to set network protocol options with `dbmlsync`, see [“CommunicationAddress \(adr\) extended option” on page 128](#).

See also

- [“Configuring MobiLink clients to use transport-layer security” \[SQL Anywhere Server - Database Administration\]](#)
- [“fips” on page 36](#)
- [“Encrypting MobiLink client/server communications” \[SQL Anywhere Server - Database Administration\]](#)
- [“-x mlsrv12 option” \[MobiLink - Server Administration\]](#)
- [“certificate_company” on page 29](#)
- [“certificate_name” on page 30](#)
- [“certificate_unit” on page 32](#)
- [“trusted_certificates” on page 53](#)

Example

The following example sets up RSA encryption for a TCP/IP protocol. This requires setup on the server and client. Each command must be written on one line.

On the server, the implementation is:

```
mlsrv12
-c "dsn=SQL Anywhere 12 Demo;uid=DBA;pwd=sql"
-x tls(
  port=9999;
  tls_type=rsa;
  identity=c:\sa12\bin32\rsaserver.id;
  identity_password=test)
```

On a SQL Anywhere client, the implementation is:

```
dbmlsync -e
"CommunicationType=tls;
CommunicationAddress=
'tls_type=rsa;
trusted_certificates=\rsaroot.crt;
certificate_name=RSA Server'"
```

In an UltraLite application written in embedded SQL in C or C++, the implementation is:

```
info.stream = "tls";
info.stream_parms = TEXT(
"tls_type=rsa;
trusted_certificates=\rsaroot.crt;
certificate_name=RSA Server");
```

trusted_certificates

Specify a file containing a list of trusted root certificates used for secure synchronization.

Separately licensed component required

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

See “[Separately licensed components](#)” [*SQL Anywhere 12 - Introduction*].

Syntax

trusted_certificates=filename

Available protocols

- TLS, HTTPS

Default

None

Remarks

When synchronization occurs through a Certicom TLS synchronization stream, the MobiLink server sends its certificate to the client, and the certificate of the entity that signed it, and so on up to a self-signed root.

The client checks that the chain is valid and that it trusts the root certificate in the chain. This feature allows you to specify which root certificates to trust.

For UltraLite clients, trusted roots can be provided to ulinit and ulload when creating the database. If the trusted_certificates parameter is provided, the trusted certificates found in the file replace those stored in the database.

For Windows and Windows Mobile, if no trusted certificates are specified, the client loads the certificates from the operating system's trusted certificate store. This certificate store is used by web browsers when they connect to secure web servers via HTTPS.

For information about how to set network protocol options with dbmlsync, see “[CommunicationAddress \(adr\) extended option](#)” on page 128.

For information about how to set network protocol options with UltraLite, see “[Network protocol options for UltraLite synchronization streams](#)” [*UltraLite - Database Management and Reference*].

See also

- “Specifying file paths in an UltraLite connection parameter” [*UltraLite - Database Management and Reference*]
- “Encrypting MobiLink client/server communications” [*SQL Anywhere Server - Database Administration*]
- “-x mlsrv12 option” [*MobiLink - Server Administration*]
- “tls_type” on page 51
- “certificate_company” on page 29
- “certificate_name” on page 30
- “certificate_unit” on page 32

Example

The following example sets up RSA encryption for an HTTPS protocol. This requires setup on the server and client. Each command must be written on one line.

The server implementation is:

```
mlsrv12
-c "dsn=SQL Anywhere 12 Demo;uid=DBA;pwd=sql"
-x https(
  port=9999;
  identity=c:\sa12\bin32\rsaserver.id;
  identity_password=test)
```

On a SQL Anywhere client, the implementation is:

```
dbmlsync
-c "dsn=mydb;uid=DBA;pwd=sql"
-e "ctp=https;
  adr='port=9999;
  trusted_certificates=c:\sa12\bin32\rsaroot.crt;
  certificate_name=RSA Server'"
```

On an UltraLite client, the implementation is:

```
info.stream = "https";
info.stream_parms = TEXT(
"port=9999;
trusted_certificates=\rsaroot.crt;
certificate_name=RSA Server");
```

url_suffix

Specify the suffix to add to the URL on the first line of each HTTP request sent during synchronization.

Syntax

`url_suffix=suffix`

The syntax of *suffix* depends on whether you are using Microsoft IIS on Windows or Apache on Linux:

Redirector	Syntax of <i>suffix</i>
IIS	The default for Windows is /ias_relay_server/server/rs_server.dll.
Apache	The default for Linux is /srv/iarelayserver.

Available protocols

- HTTP, HTTPS

Default

The default is /.

Remarks

When synchronizing through a proxy or web server, the `url_suffix` may be necessary to find the MobiLink server.

For information about how to set this option when using the Relay Server, see “[Relay Server configuration file](#)” [*Relay Server*].

For information about how to set network protocol options with dbmlsync, see “[CommunicationAddress \(adr\) extended option](#)” on page 128.

For information about how to set network protocol options with UltraLite, see “[Network protocol options for UltraLite synchronization streams](#)” [*UltraLite - Database Management and Reference*].

See also

- “[Relay Server configuration file](#)” [*Relay Server*]

version

Specify the version of HTTP to use for synchronization.

Syntax

`version=HTTP-version-number`

Available protocols

- HTTP, HTTPS

Default

The default value is **1.1**.

Remarks

This option is useful if your HTTP infrastructure requires a specific version of HTTP. Values can be **1.0** or **1.1**.

For information about how to set network protocol options with dbmlsync, see “[CommunicationAddress \(adr\) extended option](#)” on page 128.

For information about how to set network protocol options with UltraLite, see “[Network protocol options for UltraLite synchronization streams](#)” [*UltraLite - Database Management and Reference*].

Example

The following example sets the HTTP version to 1.0.

On a SQL Anywhere client, the implementation is:

```
dbmlsync -e "adr=version=1.0"
```

In an UltraLite application written in embedded SQL or C++, the implementation is:

```
synch_info.stream_parms = TEXT("version=1.0");
```

zlib_download_window_size

If you set the compression option to zlib, you can use this option to specify the compression window size for download.

Syntax

```
zlib_download_window_size=window-bits
```

Available protocols

- TCPIP, TLS, HTTP, HTTPS

Default

12 on Windows Mobile, otherwise 15

Remarks

To turn off compression for downloads, set *window-bits* to 0. Otherwise, the window size can be a value between 9 and 15 inclusive. In general, better compression rates can be achieved with a higher window size, but more memory is required.

window-bits is the base two logarithm of the window size (the size of the history buffer). The following formulas can be used to determine how much memory is used on the client for each *window-bits*:

upload (compress): memory = $2^{(\text{window-bits} + 3)}$

download (decompress): memory = $2^{(\text{window-bits})}$

To support zlib compression in UltraLite, an application must call `ULEnableZlibSyncCompression(sqlca)` and `mlczlib12.dll` must be deployed.

For information about how to set network protocol options with `dbmsync`, see [“CommunicationAddress \(adr\) extended option” on page 128](#).

For information about how to set network protocol options with UltraLite, see [“Network protocol options for UltraLite synchronization streams” \[UltraLite - Database Management and Reference\]](#).

See also

- [“compression” on page 33](#)
- [“zlib_upload_window_size” on page 57](#)

Example

The following option sets compression for upload only:

```
"compression=zlib;zlib_download_window_size=0"
```

zlib_upload_window_size

If you set the compression option to `zlib`, you can use this option to specify the compression window size for upload.

Syntax

`zlib_upload_window_size=window-bits`

Available protocols

- TCP/IP, TLS, HTTP, HTTPS

Default

12 on Windows Mobile, otherwise 15

Remarks

To turn off compression for uploads, set the window size to 0. Otherwise, the window size can be a value between 9 and 15 inclusive. In general, better compression rates can be achieved with a higher window size, but more memory is required.

window-bits is the base two logarithm of the window size (the size of the history buffer). The following formulas can be used to determine how much memory is used on the client for each *window-bits*:

upload (compress): $\text{memory} = 2^{(\text{window-size} + 3)}$

download (decompress): $\text{memory} = 2^{(\text{window-size})}$

To support zlib compression in UltraLite, an application must call `ULEnableZlibSyncCompression(sqlca)` and `mlczlib12.dll` must be deployed.

For information about how to set network protocol options with dbmlsync, see [“CommunicationAddress \(adr\) extended option” on page 128](#).

For information about how to set network protocol options with UltraLite, see [“Network protocol options for UltraLite synchronization streams” \[UltraLite - Database Management and Reference\]](#).

See also

- [“compression” on page 33](#)
- [“zlib_download_window_size” on page 56](#)

Example

The following option sets compression for download only:

```
"compression=zlib;zlib_upload_window_size=0"
```

Schema changes in remote clients

Introduction to MobiLink client schema changes

As your needs evolve, deployed remote databases may require schema changes. The most common schema changes are adding a new column to an existing table or adding a new table to the database.

Previously, schema changes that affected synchronization required a successful synchronization immediately before making the schema change. This is no longer required. In order to do this, you must use new SQL syntax to store the script version on the synchronization subscription instead of using the ScriptVersion extended option.

The SQL syntax to support this feature is as follows:

- **CREATE SYNCHRONIZATION SUBSCRIPTION statement** Use the SCRIPT VERSION clause to specify the script version to use during synchronization. See [“CREATE SYNCHRONIZATION SUBSCRIPTION statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#).
- **ALTER SYNCHRONIZATION SUBSCRIPTION statement** Use the SET SCRIPT VERSION clause to specify the script version to use during synchronization. See [“ALTER SYNCHRONIZATION SUBSCRIPTION statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#).

Associating script versions with subscriptions

Starting in version 12.0.0, new functionality greatly simplifies the process of performing schema changes to remote databases. To use this functionality, you must stop using the dbmlsync ScriptVersion extended option. Instead, you should associate your script version directly with your synchronization subscription using new clauses that have been added to the CREATE SYNCHRONIZATION SUBSCRIPTION and ALTER SYNCHRONIZATION SUBSCRIPTION statements.

When you use the new syntax, each database transaction is uploaded using the script version that was associated with the subscription at the time the transaction occurred. This makes it possible to perform a schema change that requires a script version change without synchronizing.

When using the older ScriptVersion extended option, the script version is associated with the transaction at synchronization time. As a result, you must synchronize before any schema change.

A few existing synchronization systems depend on changing the script version used by a subscription between synchronizations for reasons other than schema changes. It may not be possible to update these systems to use the new functionality.

Going forward, it is recommended that you always specify the SCRIPT VERSION clause when you create a synchronization subscription. Existing subscriptions can be upgraded by following the instructions below.

If you have an existing subscription named **my_sub** that you synchronize using the dbmlsync ScriptVersion extended option, here are the steps to associate your script version directly with your subscription.

To associate a script version with a subscription

1. Determine the script version currently being used to synchronize **my_sub**. The easiest way to do this is as follows:
 - a. Add the -v+ option to your existing dbmlsync command line and synchronize.
 - b. Look in your dbmlsync output file for a line that identifies the script version being used. Look for something similar to the following:

```
Script version: my_script_ver_1
```

2. Associate the current script version with the subscription:

```
ALTER SYNCHRONIZATION SUBSCRIPTION <sub_name>  
SET SCRIPT VERSION = <ver>
```

where <sub_name> is the name of the subscription, in this case **my_sub** and <ver> is the current script version, determined in step 1.

All transactions that occur after this point are associated with the script version.

3. Synchronize one last time using your old options. This ensures that any transactions that occurred before you completed step 2 are uploaded with the correct script version.
4. Remove the ScriptVersion extended option wherever it is specified for this subscription. The extended option may be specified on the dbmlsync command line, in a synchronization profile or associated with a subscription, publication or MobiLink user in the remote database.

For databases that contain more than one subscription, repeat the previous procedure for each subscription.

Schema upgrades for SQL Anywhere remote databases

You can change the schema of remote SQL Anywhere databases after they are deployed.

Note

If you can ensure that there are no other connections to the remote database, you can use the ALTER PUBLICATION statement manually to add new or altered tables to your publications. Otherwise, you must use the `sp_hook_dbmsync_schema_upgrade` hook to upgrade your schema.

See [“sp_hook_dbmsync_schema_upgrade” on page 217](#).

To add tables to SQL Anywhere remote databases

1. Add the associated table scripts in the consolidated database.

The same script version may be used for the remote database without the new table and the remote database with the new table. However, if the presence of the new table changes how existing tables are synchronized, then you must create a new script version, and create new scripts for all tables being synchronized with the new script version.

2. Perform a normal synchronization. Ensure that the synchronization is successful before proceeding.
3. Use the ALTER PUBLICATION statement to add the table. For example,

```
ALTER PUBLICATION your_pub
ADD TABLE table_name;
```

You can use this statement inside a `sp_hook_dbmsync_schema_upgrade` hook. See [“sp_hook_dbmsync_schema_upgrade” on page 217](#).

For more information, see [“ALTER PUBLICATION statement \[MobiLink\] \[SQL Remote\]” \[SQL Anywhere Server - SQL Reference\]](#).

4. Synchronize. Use the new script version, if required.

Changing table definitions in remote databases

Changing the number or type of columns in an existing table must be done carefully. When a MobiLink client synchronizes with a new schema, it expects scripts, such as `upload_update` or `download_cursor`, which have parameters for all columns in the remote table. An older remote database expects scripts that have only the original columns.

To alter a published table in a deployed SQL Anywhere remote database

1. At the consolidated database, create a new script version.

For more information, see [“Script versions” \[MobiLink - Server Administration\]](#).

2. For your new script version, create scripts for all tables in the publication(s) that contain the table that you want to alter and that are synchronized with the old script version.

3. Perform a normal synchronization of the remote database using the old script version. Ensure that the synchronization is successful before proceeding.
4. At the remote database, use the ALTER PUBLICATION statement to temporarily drop the table from the publication. For example,

```
ALTER PUBLICATION your_pub  
DROP TABLE table_name;
```

For more information, see [“ALTER PUBLICATION statement \[MobiLink\] \[SQL Remote\]” \[SQL Anywhere Server - SQL Reference\]](#).

You can use this statement inside a sp_hook_dbmsync_schema_upgrade hook. See [“sp_hook_dbmsync_schema_upgrade” on page 217](#).

5. At the remote database, use the ALTER TABLE statement to alter the table.

For more information, see [“ALTER TABLE statement” \[SQL Anywhere Server - SQL Reference\]](#).

6. At the remote database, use the ALTER PUBLICATION statement to add the table back into the publication.

For more information, see [“ALTER PUBLICATION statement \[MobiLink\] \[SQL Remote\]” \[SQL Anywhere Server - SQL Reference\]](#).

You can use this statement inside a sp_hook_dbmsync_schema_upgrade hook. See [“sp_hook_dbmsync_schema_upgrade” on page 217](#).

7. Synchronize with the new script version.

Schema upgrades for UltraLite remote databases

You can change the schema of a remote UltraLite database by having your existing application execute DDL.

- If you deploy a new application with a new database, you need to repopulate the UltraLite database by synchronizing with the MobiLink server.
- If you deploy a new application that contains DDL to upgrade the database, your data is preserved.
- If your existing application has a generic way to receive DDL statements, it can apply DDL to your database and your data is preserved.

It is usually impractical to have all users upgrade to the new version of the application at the same time. Therefore, you need to be able to have both versions co-existing in the field and synchronizing with a single consolidated database. You can create two or more versions of the synchronization scripts that are stored in the consolidated database and control the actions of the MobiLink server. Each version of your application can then select the appropriate set of synchronization scripts by specifying the correct version name when it initiates synchronization.

For information about UltraLite DDL, see “UltraLite SQL statements” [*UltraLite - Database Management and Reference*].

See also

- “Deploying UltraLite schema upgrades” [*UltraLite - Database Management and Reference*]

SQL Anywhere clients for MobiLink

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

SQL Anywhere clients

The following sections contain topics related to using SQL Anywhere clients with MobiLink.

Creating a remote database

Any SQL Anywhere database can be used as a remote database in a MobiLink system. All you need to do is create a publication, create a MobiLink user, create a synchronization subscription, and register the user with the consolidated database.

Note

A database that does not have a transaction log can only be used as a remote database for scripted upload and for download-only publications.

If you use the **Create Synchronization Model Wizard** to create your MobiLink client application, these objects are created for you when you deploy the model. Even then, you should understand the concepts.

To use a SQL Anywhere database as a remote database

1. Start with an existing SQL Anywhere database, or create a new one and add your tables.

See [“Publishing data” on page 67](#).
2. Create one or more publications in the remote database.

See [“Creating MobiLink users” on page 74](#).
3. Create MobiLink users in the remote database.

See [“Adding MobiLink user names to the consolidated database” on page 5](#).
4. Register users with the consolidated database.

See [“Adding MobiLink user names to the consolidated database” on page 5](#).
5. Create synchronization subscriptions in the remote database.

See [“Creating synchronization subscriptions” on page 77](#).

Deploying remote databases

To deploy SQL Anywhere remote databases, you need to create the databases and add the appropriate publications. To do this, you can customize a prototype remote database.

When deploying a starter database to multiple locations, it is safest to deploy databases that have a null remote ID. If you have synchronized the databases to prepopulate them, you can set the remote ID back to null before deployment. This method ensures that the remote ID is unique because the first time the remote database synchronizes, a unique remote ID is assigned. Alternatively, the remote ID can be set as a remote setup step, but it must be unique.

See [“Setting remote IDs” on page 65](#).

To deploy MobiLink remote databases by customizing a prototype

1. Create a prototype remote database.

The prototype database should have all the tables and publications that are needed, but not the data that is specific to each database. This information typically includes the following:

- The MobiLink user name.
- Synchronization subscriptions.
- The `global_database_id` option that provides the starting point for global autoincrement key values.

2. For each remote database, perform the following operations:

- Create a directory to hold the remote database.
- Copy the prototype remote database into the directory.

If the transaction log is held in the same directory as the remote database, the log file name does not need to be changed.

- Run a SQL script that adds the individual information to the database.

The SQL script can be a parameterized script. For information about parameterized scripts, see [“PARAMETERS statement \[Interactive SQL\]” \[SQL Anywhere Server - SQL Reference\]](#), and [“Using SQL command files” \[SQL Anywhere Server - SQL Usage\]](#).

When you use the **Create Synchronization Model Wizard** to create your MobiLink client application, you can deploy your database using a wizard. See [“Deploying synchronization models” \[MobiLink - Getting Started\]](#).

See also

- [“Deploying SQL Anywhere MobiLink clients” \[MobiLink - Server Administration\]](#)
- [“First synchronization always works” on page 66](#)

Example

The following SQL script is taken from the Contact sample. It can be found in `samples-dir\MobiLink>Contact\customize.sql`. (For information about `samples-dir`, see [“Samples directory” \[SQL Anywhere Server - Database Administration\]](#).)

```
PARAMETERS ml_userid, db_id;  
go
```

```

SET OPTION PUBLIC.global_database_id = {db_id}
go
CREATE SYNCHRONIZATION USER {ml_userid}
    TYPE 'TCPIP'
    ADDRESS 'host=localhost;port=2439'
go
CREATE SYNCHRONIZATION SUBSCRIPTION TO "DBA"."Product"
    FOR {ml_userid}
go
CREATE SYNCHRONIZATION SUBSCRIPTION TO "DBA"."Contact"
    FOR {ml_userid}
go
commit work
go

```

The following command executes the script for a remote database with data source `dsn_remote_1`:

```
dbisql -c "dsn=dsn_remote_1" read customize.sql [SSinger] [2]
```

Setting remote IDs

The remote ID uniquely identifies a remote database in a MobiLink synchronization system. When a SQL Anywhere database is created, the remote ID is null. When the database synchronizes with MobiLink, MobiLink checks for a null remote ID and if it finds one, it assigns a GUID as the remote ID. Once set, the database maintains the same remote ID unless it is manually changed.

If you are going to reference remote IDs in MobiLink event scripts or elsewhere, you may want to change the remote ID to a more meaningful name. To do this, you set the `ml_remote_id` database option for the remote database. The `ml_remote_id` option is a user-defined option that is stored in the `SYSOPTION` system table. You can change it using the `SET OPTION` statement or using the SQL Anywhere 12 plug-in to Sybase Central.

The remote ID must be unique within your synchronization system.

If you set the remote ID manually and you subsequently recreate the remote database, you must either give the recreated remote database a different name from the old one or use the `ml_reset_sync_state` stored procedure to reset the state information in the consolidated database for the remote database. See [“ml_reset_sync_state system procedure”](#) [*MobiLink - Server Administration*].

For more information about changing database options, see:

- [“SET OPTION statement”](#) [*SQL Anywhere Server - SQL Reference*]
- [“Setting database options”](#) [*SQL Anywhere Server - Database Administration*]
- [“SYSOPTION system view”](#) [*SQL Anywhere Server - SQL Reference*]

Caution

The safest time to change the remote ID is before the first synchronization. If you change it later, be sure you have performed a complete, successful synchronization just before changing the remote ID. Otherwise you may lose data and put your database into an inconsistent state.

See also

- [“Remote IDs” on page 9](#)

Example

The following SQL statement sets the remote ID to the value HR001:

```
SET OPTION PUBLIC.ml_remote_id = 'HR001'
```

Upgrading remote databases

If you install a new SQL Anywhere remote database over an older version, the synchronization progress information in the consolidated database is incorrect. You can correct this problem using the `ml_reset_sync_state` stored procedure to reset the state information for the remote database in the consolidated database. See [“ml_reset_sync_state system procedure” \[MobiLink - Server Administration\]](#).

For more information about upgrading, see [“Upgrading SQL Anywhere MobiLink clients” \[SQL Anywhere 12 - Changes and Upgrading\]](#).

Progress offsets

The progress offset is an integer value that indicates the point in time up to which all operations for the subscription have been uploaded and acknowledged. The `dbmlsync` utility uses the offset to decide what data to upload. On the remote database, the offset is stored in the `progress` column of the `SYS.ISYSSYNC` system table. On the consolidated database, the offset is stored in the `progress` column of the `ml_subscription` table.

For each remote, the remote and consolidated databases maintain an offset for every subscription. When a MobiLink user synchronizes, the offsets are confirmed for all subscriptions that are associated with the MobiLink user, even if they are not being synchronized at the time. This is required because more than one publication can contain the same data. The only exception is that `dbmlsync` does not check the progress offset of a subscription until it has attempted an upload.

If there is any disagreement between the remote and consolidated database offsets, the default behavior is to update the offsets on the remote database with values from the consolidated database and then send a new upload based on those offsets. Usually this default is appropriate. For example, it is generally appropriate when the consolidated database is restored from backup and the remote transaction log is intact, or when an upload is successful but communication failure prevented an upload acknowledgement from being sent.

Most progress offset mismatches are resolved automatically using the consolidated progress values. In the rare case that you must intervene to fix a problem with progress offsets, you can use the `dbmlsync -r` option.

For more information, see [“-r dbmlsync option” on page 116](#).

First synchronization always works

The first time you attempt to synchronize a newly created subscription, the progress offsets for the subscription are not checked against those on the consolidated database. This feature allows a remote database to be recreated and synchronized without having to delete its state information, which is maintained in the consolidated database.

The dbmsync utility detects a first synchronization when the columns in the remote database system table SYS.ISYSSYNC are as follows: the value for the **progress** column is the same as the value for the **created** column, and the value for the **log_sent** column is null.

However, when you synchronize two or more subscriptions in the same upload, and one of the subscriptions is not synchronizing for the first time, then progress offsets are checked for all subscriptions being synchronized, including the ones that are being synchronized for the first time. For example, if you specify the dbmsync -s option with two subscriptions (-s sub1, pub2), and sub1 has synchronized before but sub2 has not, then the progress offsets of both subscriptions are checked against the consolidated database values.

For more information, see:

- “ISYSSYNC system table” [[SQL Anywhere Server - SQL Reference](#)]
- “Transaction log files” on page 81

Publishing data

A publication is a database object that identifies the data that is to be synchronized. It defines the data to be uploaded, and it limits the tables that can be downloaded to. (The download is defined in the download_cursor script.)

A publication consists of one or more articles. Each article specifies a subset of a table that is to be synchronized. The subset may be the entire table or a subset of its rows and/or columns. Each article in a publication must refer to a different table.

You create a subscription to link a publication to a user.

You create publications using Sybase Central or with the CREATE PUBLICATION statement.

In Sybase Central, all publications and articles appear in the **Publications** folder.

Notes about publications

- DBA authority is required to create and drop publications.
- You cannot create two publications containing different column subsets of the same table.
- The publication determines which columns are selected, but it does not determine the order in which they are sent. Columns are always sent in the order in which they were defined in the CREATE TABLE statement.
- Each article must include all the columns in the primary key of the table that it references.

- An article can limit the columns of a table that are synchronized. Using a WHERE clause, it can also limit the rows.
- Views and stored procedures cannot be included in publications.
- Publications and subscriptions are also used by the Sybase message-based replication technology, SQL Remote. SQL Remote requires publications and subscriptions in both the consolidated and remote databases. In contrast, MobiLink publications appear only in SQL Anywhere remote databases. MobiLink consolidated databases are configured using synchronization scripts.

See also

- [“CREATE PUBLICATION statement \[MobiLink\] \[SQL Remote\]” \[SQL Anywhere Server - SQL Reference\]](#)

Publishing whole tables

The simplest publication you can make consists of a set of articles, each of which contains all the rows and columns in one table. These tables must already exist.

To publish one or more entire tables (Sybase Central)

1. Connect to the remote database as a user with DBA authority, using the SQL Anywhere 12 plug-in.
2. Open the **Publications** folder.
3. Choose **File » New » Publication**.
4. In the **What Do You Want To Name The New Publication** field, enter a name for the new publication. Click **Next**.
5. Click **Next**.
6. On the **Available Tables** list, select a table. Click **Add**.
7. Click **Finish**.

To publish one or more entire tables (SQL)

1. Connect to the remote database as a user with DBA authority.
2. Execute a CREATE PUBLICATION statement that specifies the name of the new publication and the table you want to publish.

See [“CREATE PUBLICATION statement \[MobiLink\] \[SQL Remote\]” \[SQL Anywhere Server - SQL Reference\]](#).

Example

The following statement creates a publication that publishes the whole Customers table:

```
CREATE PUBLICATION pub_customer (  
  TABLE Customers  
)
```

The following statement creates a publication including all columns and rows in each of a set of tables from the SQL Anywhere sample database:

```
CREATE PUBLICATION sales (  
  TABLE Customers,  
  TABLE SalesOrders,  
  TABLE SalesOrderItems,  
  TABLE Products  
)
```

Publishing only some columns in a table

You can create a publication that contains all the rows, but only some of the columns of a table from Sybase Central or by listing the columns in the CREATE PUBLICATION statement.

Note

- If you create two publications that include the same table with different column subsets, then you may only create a synchronization subscription for one of them.
- An article must include all the primary key columns in the table.

To publish only some columns in a table (Sybase Central)

1. Connect to the remote database as a user with DBA authority using the SQL Anywhere 12 plug-in.
2. Open the **Publications** folder.
3. Choose **File » New » Publication**.
4. In the **What Do You Want To Name The New Publication** field, enter a name for the new publication. Click **Next**.
5. Click **Next**.
6. On the **Available Tables** list, select a table. Click **Add**.
7. Click **Next**.
8. In the **Available Columns** list, expand the list of available columns. Select a column and click **Add**.
9. Click **Finish**.

To publish only some columns in a table (SQL)

1. Connect to the remote database as a user with DBA authority.
2. Execute a CREATE PUBLICATION statement that specifies the publication name and the table name. List the published columns in parenthesis following the table name.

See “[CREATE PUBLICATION statement \[MobiLink\] \[SQL Remote\]](#)” [[SQL Anywhere Server - SQL Reference](#)].

Example

The following statement creates a publication that publishes all rows of the id, company_name, and city columns of the Customers table:

```
CREATE PUBLICATION pub_customer (  
  TABLE Customers (id, company_name,  
    city )  
)
```

Publishing only some rows in a table

When no WHERE clause is specified in an article definition, all changed rows in the table are uploaded. You can add WHERE clauses to articles in the publication to limit the rows to be uploaded to those that have changed and that satisfy the search condition in the WHERE clause.

The search condition in the WHERE clause can only reference columns that are included in the article. In addition, you cannot use any of the following in the WHERE clause:

- subqueries
- variables
- non-deterministic functions

These conditions are not enforced, but breaking them can lead to unexpected results. Any errors relating to the WHERE clause are generated when the DML is run against the table referred to by the WHERE clause, and not when the publication is defined.

To create a publication using a WHERE clause (Sybase Central)

1. Connect to the remote database as a user with DBA authority using the SQL Anywhere 12 plug-in.
2. Open the **Publications** folder.
3. Choose **File » New » Publication**.
4. In the **What Do You Want To Name The New Publication** field, enter a name for the new publication. Click **Next**.

5. Click **Next**.
6. On the **Available Tables** list, select a table. Click **Add**.
7. Click **Next**.
8. Click **Next**.
9. In the **Articles List**, select a table and enter the search condition in the **The Selected Article Has the following WHERE clause** pane.
10. Click **Finish**.

To create a publication using a WHERE clause (SQL)

1. Connect to the remote database as a user with DBA authority.
2. Execute a CREATE PUBLICATION statement that includes the tables you want to include in the publication and a WHERE condition.

See “[CREATE PUBLICATION statement \[MobiLink\] \[SQL Remote\]](#)” [[SQL Anywhere Server - SQL Reference](#)].

Example

The following example creates a publication that includes the entire Employees table and all rows in the SalesOrders table that have not been marked as archived.

```
CREATE PUBLICATION main_publication (  
  TABLE Employees,  
  TABLE SalesOrders  
  WHERE archived = 'N'  
);
```

By changing the archived column in the table from any other value to an N, a delete is sent to the MobiLink server during the next synchronization. Conversely, by changing the archived column from N to any other value, an insert is sent. The update to the archived column is *not* sent to the MobiLink server.

Download-only publications

You can create a publication that only downloads data to remote databases, and never uploads data. Download-only publications do not use a transaction log on the client.

Differences between download-only methods

There are two ways to specify that only a download (and not an upload) should occur:

- **Download-only synchronization** Use the dbmlsync options -e DownloadOnly or -ds.
- **Download-only publication** Create the publication with the FOR DOWNLOAD ONLY keyword.

The two approaches are quite different:

Download-only synchronizations	Download-only publications
If the download attempts to change rows that have been modified on the remote database and not yet uploaded, the download fails.	The download can overwrite rows that have been modified on the remote database and not yet uploaded.
Uses a normal publication that can be uploaded and/or downloaded. Download-only synchronization is specified using dbmlsync command line options or extended options.	Uses a download-only publication. All synchronizations on these publications are download-only. You cannot alter a normal publication to make it download-only.
Requires a log file.	Does not require a log file.
The log file is not truncated when these subscriptions are not uploaded for a long time, and can consume significant amounts of storage.	If there is a log file, the synchronization does not affect the synchronization truncation point. This means that the log file can still be truncated even if the publication is not synchronized for a long time. Download-only publications do not affect log file truncation.
You need to do an upload occasionally to reduce the amount of log that is scanned by the download-only synchronization. Otherwise, the download-only synchronization takes an increasingly long time to complete.	There is no need to ever do an upload.

See also

- [“CREATE PUBLICATION statement \[MobiLink\] \[SQL Remote\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“Upload-only and download-only synchronizations” \[MobiLink - Server Administration\]](#)

Altering existing publications

After you have created a publication, you can alter it by adding, modifying, or deleting articles, or by renaming the publication. If an article is modified, the entire specification of the modified article must be entered.

You can perform these tasks using Sybase Central or with the ALTER PUBLICATION statement.

Notes

- Publications can be altered only by the DBA or the publication's owner.
- Be careful. In a running MobiLink setup, altering publications may cause errors and can lead to loss of data. If the publication you are altering has any subscriptions, then you must treat this change as a schema upgrade. See [“Schema changes in remote clients” on page 58](#).

To modify the properties of existing publications or articles (Sybase Central)

1. Connect to the remote database as a user who owns the publication or as a user with DBA authority.
2. In the left pane, click the publication or article. The properties appears in the right pane.
3. Configure the properties.

To add articles (Sybase Central)

1. Connect to the remote database as a user who owns the publication or as a user with DBA authority using the SQL Anywhere 12 plug-in.
2. Expand the **Publications** folder.
3. Select a publication.
4. Choose **File » New » Article**.
5. In the **Create Article Wizard**, do the following:
 - In the **Which Table Do You Want To Use For This Article** list, select a table. Click **Next**.
 - Click **Selected Columns** and select the columns. Click **Next**.
 - In the **You Can Specify a WHERE Clause For This Article** pane, enter an optional WHERE clause. Click **Finish**.

To remove articles (Sybase Central)

1. Connect to the database as a user who owns the publication or as a user with DBA authority using the SQL Anywhere 12 plug-in.
2. Expand the **Publications** folder.
3. Right-click the publication and choose **Delete**.
4. Click **Yes**.

To modify an existing publication (SQL)

1. Connect to the remote database as a user who owns the publication or as a user with DBA authority.
2. Execute an ALTER PUBLICATION statement.

See [“ALTER PUBLICATION statement \[MobiLink\] \[SQL Remote\]” \[SQL Anywhere Server - SQL Reference\]](#).

Example

- The following statement adds the Customers table to the pub_contact publication.

```
ALTER PUBLICATION pub_contact  
ADD TABLE Customers
```

See also the “[ALTER PUBLICATION statement \[MobiLink\] \[SQL Remote\]](#)” [*SQL Anywhere Server - SQL Reference*].

Dropping publications

You can drop a publication using either Sybase Central or the DROP PUBLICATION statement.

You must have DBA authority to drop a publication or be the owner of the publication.

To delete a publication (Sybase Central)

1. Connect to the remote database as a user with DBA authority using the SQL Anywhere 12 plug-in.
2. Open the **Publications** folder.
3. Right-click a publication and choose **Delete**.

To delete a publication (SQL)

1. Connect to the remote database as a user with DBA authority.
2. Execute a DROP PUBLICATION statement.

See “[DROP PUBLICATION statement \[MobiLink\] \[SQL Remote\]](#)” [*SQL Anywhere Server - SQL Reference*].

Example

The following statement drops the publication named pub_orders.

```
DROP PUBLICATION pub_orders
```

Creating MobiLink users

A MobiLink user name is used to authenticate when you connect to the MobiLink server. You must create MobiLink users in the remote database, and then register them on the consolidated database.

MobiLink users are not the same as database users. You can create a MobiLink user name that matches the name of a database user, but neither MobiLink nor SQL Anywhere is affected by this coincidence.

To add a MobiLink user to a remote database (Sybase Central)

1. Connect to the database from the SQL Anywhere 12 plug-in as a user with DBA authority.
2. Click the **MobiLink Users** folder.

3. Choose **File » New » MobiLink User**.
4. In the **What Do You Want To Name The New MobiLink User** field, enter a name for the MobiLink user.
5. Click **Finish**.

To add a MobiLink user to a remote database (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a CREATE SYNCHRONIZATION USER statement. The MobiLink user name uniquely identifies a remote database and so must be unique within your synchronization system.

The following example adds a MobiLink user named SSinger:

```
CREATE SYNCHRONIZATION USER SSinger
```

You can specify properties for the MobiLink user as part of the CREATE SYNCHRONIZATION USER statement, or you can specify them separately with an ALTER SYNCHRONIZATION USER statement.

For more information, see [“CREATE SYNCHRONIZATION USER statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#).

For information about setting MobiLink user properties, including the password, see [“Storing extended options for MobiLink users” on page 75](#).

For information about registering MobiLink users, see [“Adding MobiLink user names to the consolidated database” on page 5](#).

Storing extended options for MobiLink users

You can specify options for each MobiLink user in the remote database by using extended options. Extended options can be specified on the command line, stored in the database, or specified with the sp_hook_dbmlsync_set_extended_options event hook.

For a list of extended options, see [“MobiLink SQL Anywhere client extended options” on page 126](#).

To store MobiLink extended options in the database (Sybase Central)

1. Connect to the database from the SQL Anywhere 12 plug-in as a user with DBA authority.
2. Open the MobiLink **Users** folder.
3. Right-click the MobiLink user name and choose **Properties**.
4. Change the properties as needed.

To store MobiLink extended options in the database (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute an ALTER SYNCHRONIZATION USER statement.

The following example changes the extended options for MobiLink user named SSinger to their default values:

```
ALTER SYNCHRONIZATION USER SSinger
DELETE ALL OPTION
```

For more information, see “ALTER SYNCHRONIZATION USER statement [MobiLink]” [[SQL Anywhere Server - SQL Reference](#)].

You can also specify properties when you create the MobiLink user name.

For more information, see “CREATE SYNCHRONIZATION USER statement [MobiLink]” [[SQL Anywhere Server - SQL Reference](#)].

To specify MobiLink user properties with a client event hook

- You can programmatically customize the behavior of an upcoming synchronization.

For more information, see “[sp_hook_dbmlsync_set_extended_options](#)” on page 219.

See also

- “[Using dbmlsync extended options](#)” on page 80

Dropping MobiLink users

To drop a MobiLink user from a remote database (Sybase Central)

1. Connect to the database from the SQL Anywhere 12 plug-in as a user with DBA authority.
2. Locate the MobiLink user in the MobiLink **Users** folder.
3. Right click the MobiLink user and choose **Delete**.

To drop a MobiLink user from a remote database (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a DROP SYNCHRONIZATION USER statement.

The following example removes the MobiLink user named SSinger from the database:

```
DROP SYNCHRONIZATION USER SSinger
```

For more information, see [“DROP SYNCHRONIZATION USER statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#).

Creating synchronization subscriptions

After creating MobiLink users and publications, you must subscribe at least one MobiLink user to one or more pre-existing publications. You do this by creating synchronization subscriptions.

For information about creating publications, see [“Publishing data” on page 67](#). For information about creating MobiLink users, see [“Creating MobiLink users” on page 74](#).

Note

You must ensure that all subscriptions for a MobiLink user are synchronized to only one consolidated database. Otherwise, you may experience data loss and unpredictable behavior.

A synchronization subscription links a particular MobiLink user with a publication. It can also contain other information needed for synchronization. For example, you can specify the address of the MobiLink server and options for a synchronization subscription. Values for a specific synchronization subscription override those set for MobiLink users.

Synchronization subscriptions are required only in MobiLink SQL Anywhere remote databases. Server logic is implemented through synchronization scripts, stored in the MobiLink system tables in the consolidated database.

A single SQL Anywhere database can synchronize with more than one MobiLink server. To allow synchronization with multiple servers, create different MobiLink users for each server.

See [“CREATE SYNCHRONIZATION SUBSCRIPTION statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#).

Example

To synchronize the Customers and SalesOrders tables in the SQL Anywhere sample database, you could use the following statements.

1. First, create a publication containing the Customers and SalesOrders tables. Give the publication the name testpub.

```
CREATE PUBLICATION testpub
(TABLE Customers, TABLE SalesOrders)
```

2. Next, create a MobiLink user. In this case, the MobiLink user is demo_ml_user.

```
CREATE SYNCHRONIZATION USER demo_ml_user
```

3. To complete the process, create a synchronization subscription named my_sub that links the user and the publication.

```
CREATE SYNCHRONIZATION SUBSCRIPTION my_sub TO testpub
FOR demo_ml_user
```

```
TYPE tcpip
ADDRESS 'host=localhost;port=2439;'
SCRIPT VERSION 'version1'
```

Altering MobiLink subscriptions

Synchronization subscriptions can be altered using Sybase Central or the ALTER SYNCHRONIZATION SUBSCRIPTION statement. The syntax is similar to that of the CREATE SYNCHRONIZATION SUBSCRIPTION statement, but provides an extension to more conveniently add, modify, and delete options.

To alter a synchronization subscription (Sybase Central)

1. Connect to the database as a user with DBA authority.
2. Open the MobiLink **Users** folder.
3. Click a user. The properties appear in the right pane.
4. In the right pane, click the **Subscriptions** tab. Right-click the subscription you want to change and select **Properties**.
5. Change the properties as needed.

To alter a synchronization subscription (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute an ALTER SYNCHRONIZATION SUBSCRIPTION statement.

See “ALTER SYNCHRONIZATION USER statement [MobiLink]” [[SQL Anywhere Server - SQL Reference](#)].

Dropping MobiLink subscriptions

You can delete a synchronization subscription using either Sybase Central or the DROP SYNCHRONIZATION SUBSCRIPTION statement.

You must have DBA authority to drop a synchronization subscription.

To delete a synchronization subscription (Sybase Central)

1. Connect to the database as a user with DBA authority.
2. Open the MobiLink **Users** folder.
3. Select a MobiLink user.

4. Right-click a subscription and choose **Delete**.

To delete a synchronization subscription (SQL)

1. Connect to the database as a user with DBA authority.
2. Execute a DROP SYNCHRONIZATION SUBSCRIPTION statement.

Example

The following statement drops the synchronization subscription named my_sub.

```
DROP SYNCHRONIZATION SUBSCRIPTION my_sub
```

See “[DROP SYNCHRONIZATION SUBSCRIPTION statement \[MobiLink\]](#)” [*SQL Anywhere Server - SQL Reference*].

Initiating synchronization

The client always initiates MobiLink synchronization. For SQL Anywhere clients, synchronization can be initiated using the dbmlsync utility, the dbmlsync API or the SQL SYNCHRONIZE statement. All share similar semantics but offer different interfaces to synchronization and different abilities to integrate synchronization with your own applications.

See: .

- “[MobiLink SQL Anywhere client utility \(dbmlsync\)](#)” on page 93
- “[Dbmlsync API](#)” on page 90
- “[SYNCHRONIZE statement \[MobiLink\]](#)” [*SQL Anywhere Server - SQL Reference*]

There are many options available to customize synchronization behavior, however, a few stand out because they are required for virtually any synchronization. These are discussed below.

The -c option lets you specify connection parameters that control how dbmlsync will connect to the remote database. This information is not required when using the SQL synchronize statement because the connection information is taken from the database connection that is executing the statement. See “[-c dbmlsync option](#)” on page 101.

The -s or Subscription option allows you to specify which subscription defined in the remote database will be synchronized. See “[-s dbmlsync option](#)” on page 117.

The CommunicationAddress and CommunicationType extended options let you specify network protocol options that determine how dbmlsync will connect with the MobiLink server during synchronization. See “[CommunicationAddress \(adr\) extended option](#)” on page 128 and “[CommunicationType \(ctp\) extended option](#)” on page 129.

The Script Version clause on the CREATE SYNCHRONIZATION SUBSCRIPTION SQL statement lets you specify the script version to be used when synchronizing a subscription. The script version determines which scripts will be used by the MobiLink Server to control and process the synchronization.

See [“CREATE SYNCHRONIZATION SUBSCRIPTION statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#).

Permissions for dbmsync

To synchronize, dbmsync must connect to the remote database with a user ID and password that gives it DBA authority.

You may wish to give an individual the ability to synchronize but not want to give them DBA authority over the database. You can do this by granting REMOTE DBA authority to a user ID that does not have DBA authority. A user ID with REMOTE DBA authority has DBA authority only when the connection is made from the dbmsync utility. Any other connection using the same user ID is granted no special authority.

You can give the REMOTE DBA user ID and password to whoever you wish to synchronize the database. It will allow them to synchronize but not give them any other special authority over the database.

See [“GRANT REMOTE DBA statement \[MobiLink\] \[SQL Remote\]” \[SQL Anywhere Server - SQL Reference\]](#).

Customizing synchronization

See [“Customizing dbmsync synchronization” on page 89](#).

Using dbmsync extended options

MobiLink provides several extended options to customize the synchronization process. Extended options can be set for publications, users, and subscriptions. In addition, extended option values can be overridden using options on the dbmsync command line or the `sp_hook_dbmsync_set_extended_options` hook procedure. See [“sp_hook_dbmsync_set_extended_options” on page 219](#)

For a complete list of extended options, see [“MobiLink SQL Anywhere client extended options” on page 126](#).

To override an extended option on the dbmsync command line

- Supply the extended option values in the `-e` or `-eu` dbmsync options for dbmsync, in the form `option-name=value`. For example:

```
dbmsync -e "v=on;sc=low"
```

To set an extended option for a subscription, publication or user

- Add the option to the CREATE SYNCHRONIZATION SUBSCRIPTION statement or CREATE SYNCHRONIZATION USER statement in the SQL Anywhere remote database.

Adding an extended option for a publication is a little different. To add an extended option for a publication, use the ALTER/CREATE SYNCHRONIZATION SUBSCRIPTION statement and omit the FOR clause.

Example

The following statement creates a synchronization subscription that uses extended options to set the cache size for preparing the upload to 3 MB and the upload increment size to 3 KB.

```
CREATE SYNCHRONIZATION SUBSCRIPTION TO my_pub
FOR ml_user
ADDRESS 'host=test.internal;port=2439;'
OPTION memory='3m',increment='3k'
```

Note that the option values must be enclosed in single quotes, but the option names must remain unquoted.

Dbmlsync network protocol options

Dbmlsync connection information includes the protocol to use for communications with the server, the address for the MobiLink server, and other connection parameters.

For more information, see:

- [“CommunicationType \(ctp\) extended option” on page 129](#)
- [“CommunicationAddress \(adr\) extended option” on page 128](#)

Transaction log files

Usually, dbmlsync determines what to upload by using the SQL Anywhere transaction log.

SQL Anywhere databases maintain transaction logs by default. You can determine where the transaction log is located, or whether to have one, when you create the database or afterwards using the `dblog` utility.

The transaction log is not be required to synchronize scripted upload publications or only use download-only publications.

To prepare the upload, the dbmlsync utility requires access to all transaction logs written since the last successful synchronization of all subscriptions for the MobiLink user who is synchronizing. However, SQL Anywhere log files are typically truncated and renamed as part of regular database maintenance. In such a case, old log files must be renamed and saved in a separate directory until all changes they describe have been synchronized successfully.

You can specify the directory that contains the renamed log files on the dbmlsync command line. You may omit this parameter if the working log file has not been truncated and renamed since you last synchronized, or if you run dbmlsync from the directory that contains the renamed log files.

See also

- [“Backup and data recovery” \[SQL Anywhere Server - Database Administration\]](#)
- [“Progress offsets” on page 66](#)
- [“The transaction log” \[SQL Anywhere Server - Database Administration\]](#)
- [“Initialization utility \(dbinit\)” \[SQL Anywhere Server - Database Administration\]](#)
- [“Scripted upload” on page 310](#)

Example

Suppose that the old log files are stored in the directory `c:\oldlogs`. You could use the following command to synchronize the remote database.

```
dbmsync -c "dbn=remote;uid=syncuser" c:\oldlogs
```

The path to the old logs directory must be the final argument on the command line.

Concurrency during synchronization

To ensure the integrity of synchronizations, `dbmsync` must ensure that no changes downloaded from the server modify rows in the remote that have been changed since the last upload was sent. By default, it usually does this without locking any tables so the impact on other concurrent users of the database is minimized. Tables are locked IN SHARE MODE when synchronizing a publication that uses scripted upload or when the `sp_hook_dbmsync_schema_upgrade` hook is defined.

When tables are not locked, `dbmsync` tracks all rows that are modified after the upload is built. If the download contains a change for one of these rows that is considered a conflict.

If a conflict is detected, the download phase is canceled and the download operations rolled back to avoid overwriting the new change. The `dbmsync` utility then retries the synchronization, including the upload step. This time, because the row is present at the beginning of the synchronization process, it is included in the upload and therefore not lost.

By default, `dbmsync` retries synchronization until success is achieved. You can limit the number of retries using the extended option `ConflictRetries`. Setting `ConflictRetries` to `-1` causes `dbmsync` to retry until success is achieved. Setting it to a non-negative integer causes `dbmsync` to retry for not more than the specified number of times.

For more information, see `ConflictRetries (cr) extended option` [“ConflictRetries \(cr\) extended option” on page 129](#). -----

-d option

When using the locking mechanism, if other connections to the database exist and if these connections have any locks on the synchronization tables, then synchronization fails. If you want to ensure that synchronization proceeds immediately even if other locks exist, use the `dbmsync -d` option. When this option is specified, any connections with locks that would interfere with synchronization are dropped by the database so that synchronization can proceed. Uncommitted changes on the dropped connections are rolled back.

For more information, see [“-d dbmsync option” on page 103](#).

LockTables option

You can force `dbmsync` to lock tables during synchronization using the `LockTables` extended option. You may find it desirable to lock tables during synchronization to simplify logic that you write in hook procedures.

For more information, see [“LockTables \(lt\) extended option” on page 138](#).

Initiating synchronization from an application

You may want to include the features of dbmsync in your application, rather than provide a separate executable to your remote users.

There are three ways to do this:

- Dbmsync API
For more information, see [“Dbmsync API” on page 90](#).
- SQL SYNCHRONIZE statement
For more information, see [“SYNCHRONIZE statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#).
- If you are developing in any language that can call a DLL, then you can access dbmsync through the DBTools interface. If you are programming in C or C++, you can include the *dbtools.h* header file, located in the *SDK\Include* subdirectory of your SQL Anywhere 12 directory. This file contains a description of the *a_sync_db* structure and the *DBSynchronizeLog* function, which you use to add this functionality to your application. This solution works on all supported platforms, including Windows and Unix.

The Dbmsync API and the SQL SYNCHRONIZE statement are both easier to use than the DBTools interface and you are strongly encouraged to consider using them first.

For more information, see:

- [“DBTools interface for dbmsync” on page 304](#)
- [“DBSynchronizeLog function” \[SQL Anywhere Server - Programming\]](#)
- [“a_sync_db structure” \[SQL Anywhere Server - Programming\]](#)

Using Microsoft ActiveSync synchronization

Microsoft ActiveSync is synchronization software for Microsoft Windows Mobile handheld devices. Microsoft ActiveSync governs synchronization between a Windows Mobile device and a desktop computer. A MobiLink provider for Microsoft ActiveSync governs synchronization to the MobiLink server.

Setting up Microsoft ActiveSync synchronization for SQL Anywhere clients involves the following steps:

- Configure the SQL Anywhere remote database for Microsoft ActiveSync synchronization.
See [“Configuring SQL Anywhere remote databases for Microsoft ActiveSync” on page 84](#).
- Install the MobiLink provider for Microsoft ActiveSync.
See [“Installing the MobiLink provider for Microsoft ActiveSync” on page 84](#).
- Register the SQL Anywhere client for use with Microsoft ActiveSync.
See [“Registering SQL Anywhere clients for Microsoft ActiveSync” on page 86](#).

If you use Microsoft ActiveSync synchronization, synchronization must be initiated from the Microsoft ActiveSync software. The MobiLink provider for Microsoft ActiveSync can start dbmlsync or it can wake a dbmlsync that is sleeping as scheduled by a schedule string.

You can also put dbmlsync into a sleep mode using a delay hook in the remote database, but the MobiLink provider for Microsoft ActiveSync cannot invoke synchronization from this state.

For information about scheduling synchronization, see [“Scheduling synchronization” on page 87](#).

Configuring SQL Anywhere remote databases for Microsoft ActiveSync

To configure your SQL Anywhere remote database for Microsoft ActiveSync

1. Select a synchronization type (TCP/IP, TLS, HTTP, or HTTPS).

The synchronization type can be set for a synchronization publication, for a synchronization user, or for a synchronization subscription. It is set in a similar manner for each. Here is part of a typical CREATE SYNCHRONIZATION USER statement:

```
CREATE SYNCHRONIZATION USER SSinger
TYPE tcpip
...
```

2. Supply an address clause to specify communication between the MobiLink provider for Microsoft ActiveSync and the MobiLink server.

For HTTP or TCP/IP synchronization, the ADDRESS clause of the CREATE SYNCHRONIZATION USER or CREATE SYNCHRONIZATION SUBSCRIPTION statement specifies communication between the MobiLink client and server. For Microsoft ActiveSync, the communication takes place in two stages: from the dbmlsync utility on the device to the MobiLink provider for Microsoft ActiveSync on the desktop computer, and from the desktop computer to the MobiLink server. The ADDRESS clause specifies the communication between MobiLink provider for Microsoft ActiveSync and the MobiLink server.

The following statement specifies TCP/IP communication to a MobiLink server on a computer named kangaroo:

```
CREATE SYNCHRONIZATION USER SSinger
TYPE tcpip
ADDRESS 'host=kangaroo;port=2439'
```

For more information, see [“CREATE SYNCHRONIZATION USER statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#).

Installing the MobiLink provider for Microsoft ActiveSync

Before you register your SQL Anywhere MobiLink client for use with Microsoft ActiveSync, you must install the MobiLink provider for Microsoft ActiveSync using the installation utility (*mlasinst.exe*).

The SQL Anywhere for Windows Mobile installer installs the MobiLink provider for Microsoft ActiveSync. If you install SQL Anywhere for Windows Mobile you do not need to perform the steps in this section.

When you have installed the MobiLink provider for Microsoft ActiveSync you must register each application separately. For instructions, see [“Registering SQL Anywhere clients for Microsoft ActiveSync” on page 86](#).

To install the MobiLink provider for Microsoft ActiveSync

1. Ensure that you have the Microsoft ActiveSync software on your computer, and that the Windows Mobile device is connected.
2. Run the following command to install the MobiLink provider:

```
mlasinst -k desk-path -v dev-path
```

where *desk-path* is the location of the desktop component of the provider (*mlasdesk.dll*) and *dev-path* is the location of the device component (*mlasdev.dll*).

If you have SQL Anywhere installed on your computer, *mlasdesk.dll* is located in *install-dir\bin32*; *mlasdev.dll* is located in *install-dir\CE*. If you omit *-v* or *-k*, these directories are searched by default.

If you receive a message telling you that the remote provider failed to open, perform a soft reset of the device and repeat the command:

For more information, see [“Microsoft ActiveSync Provider Installation utility \(mlasinst\)” on page 18](#).

3. Restart your computer.

Microsoft ActiveSync does not recognize new providers until the computer is restarted.

4. Enable the MobiLink provider.

For Windows versions before Vista:

- In the Microsoft ActiveSync window, click **Options**.
- Check the MobiLink item in the list and click **OK** to activate the provider.
- To see a list of registered applications, click **Options** again, choose the MobiLink provider, and click **Settings**.

For more information about registering applications, see [“Registering SQL Anywhere clients for Microsoft ActiveSync” on page 86](#).

For Windows Vista:

- From the Windows Mobile Device Center window, click **Mobile Device Settings** and then click **Change Content Settings**.

- Select **MobiLink Clients** and click **Save** to activate the provider.
- To see a list of registered applications, click **Change Content Settings**, click **MobiLink Clients**, and then click **Sync Settings**.

Registering SQL Anywhere clients for Microsoft ActiveSync

You can register your application for use with Microsoft ActiveSync either by using the Microsoft ActiveSync provider install utility or using the Microsoft ActiveSync software itself. This section describes how to use the Microsoft ActiveSync software.

For information about the alternative approach, see [“Microsoft ActiveSync Provider Installation utility \(mlasinst\)” on page 18](#).

To register the SQL Anywhere client for use with Microsoft ActiveSync

1. Ensure that the MobiLink provider for Microsoft ActiveSync is installed.

For information, see [“Installing the MobiLink provider for Microsoft ActiveSync” on page 84](#).

2. Start the Microsoft ActiveSync software on your desktop computer.
3. For Windows versions before Vista:
 - From the **Microsoft ActiveSync** window, choose **Options**.
 - From the list of information types, choose **MobiLink** and click **Settings**.
 - In the **MobiLink Synchronization** window, click **New**.

For Windows Vista:

- From the **Windows Mobile Device Center** window, click **Mobile Device Settings** and then click **Change Content Settings**.
 - Click **Change Content Settings**.
 - Click **MobiLink Clients**.
 - Click **Sync Settings**.
4. Enter the following information for your application:
 - **Application name** A name identifying the application to be displayed in the Microsoft ActiveSync user interface.
 - **Class name** The class name for the dbmlsync client, as set using the -wc option.

For more information, see [“-wc dbmlsync option” on page 124](#).

- **Path** The location of the dbmlsync application on the device.
- **Arguments** Any command line arguments to be used when Microsoft ActiveSync starts dbmlsync.

You start dbmsync in one of two modes:

- If you specify scheduling options, dbmsync enters hover mode. In this case, use the dbmsync -wc option with a matching value in the class name setting.

For more information, see [“-wc dbmsync option” on page 124](#) and [“Scheduling synchronization” on page 87](#).

- Otherwise, dbmsync is not in hovering mode. In this case, use -k to shut down dbmsync.

For more information, see [“-k dbmsync option \(deprecated\)” on page 109](#).

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

Scheduling synchronization

You can set up dbmsync to synchronize periodically based on rules you define. There are two ways you can set this up:

- Use the dbmsync extended option SCHEDULE to initiate synchronization at specific times of the day or week or at regular intervals. In this case, dbmsync remains running until stopped by the user.

See [“Setting up scheduling with dbmsync options” on page 88](#).

- Use dbmsync event hooks to initiate synchronization based on logic that you define. This is the best way to implement synchronization at irregular intervals or in response to an event. In this case, you can stop dbmsync programmatically from your hook code.

See [“Initiating synchronization with event hooks” on page 88](#).

This method is not available when the Dbmsync API or the SQL SYNCHRONIZE statement is used.

Hovering

When hovering, dbmsync scans the database transaction log and builds its upload during the delay between synchronizations. This allows synchronization to proceed more quickly when it is triggered because some of the work is already done.

When hovering, dbmsync scans to the end of the transaction log then polls the log periodically for new transactions. You can control the interval between polls using the PollingPeriod extended option of the -pp option. See [“PollingPeriod \(pp\) extended option” on page 142](#).

When you are hovering on two or more subscriptions at the same time, you can use the HoverRescanThreshold extended option or the sp_hook_dbmsync_log_rescan event hook to limit memory usage by recovering otherwise lost memory.

Hovering can be disabled using the DisablePolling extended option or the -p option.

For more information, see:

- [“HoverRescanThreshold \(hrt\) extended option” on page 135](#)
- [“-p dbmsync option” on page 112](#)
- [“DisablePolling \(p\) extended option” on page 131](#)
- [“sp_hook_dbmsync_log_rescan” on page 205](#)

Setting up scheduling with dbmsync options

Instead of running dbmsync in a batch fashion, where it synchronizes and then shuts down, you can set up a SQL Anywhere client so that dbmsync runs continuously, synchronizing at predetermined times.

You specify the synchronization schedule as an extended option. It can be specified either on the dbmsync command line or it can be stored in the database for the synchronization user, subscription, or publication.

This method is not available when the Dbmsync API or the SQL SYNCHRONIZE statement is used.

For more information about scheduling syntax, see [“Schedule \(sch\) extended option” on page 143](#).

For more information about extended options, see:

- [“MobiLink SQL Anywhere client extended options” on page 126](#)
- [“-eu dbmsync option” on page 108](#)

To add scheduling to the synchronization subscription

- Set the Schedule extended option in the synchronization subscription. For example,

```
CREATE SYNCHRONIZATION SUBSCRIPTION TO mypub
FOR mluser
ADDRESS 'host=localhost'
OPTION schedule='weekday@11:30am-12:30pm'
```

You can override scheduling and synchronize immediately using the dbmsync -is option. The -is option instructs dbmsync to ignore scheduling that is specified with the scheduling extended option. For more information, see [“-is dbmsync option” on page 108](#).

To add scheduling from the dbmsync command line

- Set the schedule extended option. Extended options are set with -e or -eu. For example,

```
dbmsync -e "sch=weekday@11:30am-12:30pm" ...
```

If scheduled synchronization is specified in either place, dbmsync does not shut down after synchronizing, but runs continuously.

Initiating synchronization with event hooks

There are dbmsync event hooks that you can implement to control when synchronization occurs.

With the `sp_hook_dbmlsync_end` hook, you can use the Restart row in the `#hook_dict` table to decide at the end of each synchronization if `dbmlsync` should repeat the synchronization.

For more information, see [“sp_hook_dbmlsync_end” on page 203](#).

With the `sp_hook_dbmlsync_delay` hook you can create a delay at the beginning of each synchronization that allows you to choose the time to proceed with synchronization. With this hook it is possible to delay for a fixed amount of time or to poll periodically, waiting for some condition to be satisfied.

For more information, see [“sp_hook_dbmlsync_delay” on page 191](#).

This method is not available when the `Dbmlsync` API or the SQL `SYNCHRONIZE` statement is used.

Customizing dbmlsync synchronization

dbmlsync client event hooks

Event hooks allow you to use SQL stored procedures to manage the client-side synchronization process for `dbmlsync`. You can use client event hooks with the `dbmlsync` command line utility or the `dbmlsync` programming interfaces.

You can use event hooks to log and handle synchronization events. For example, you can schedule synchronizations based on logical events, retry connection failures, or handle specific errors and referential integrity violations.

For more information about client event hooks, see [“Event hooks for SQL Anywhere clients” on page 178](#).

dbmlsync programming interfaces

You can use the following programming interfaces to integrate MobiLink clients into your applications and start synchronizations. These interfaces provide an alternative to the `dbmlsync` command line utility.

- **dbmlsync API** The `Dbmlsync` API provides a programming interface that allows MobiLink clients written in C++ or .Net to launch synchronizations and receive feedback about the progress of the synchronizations they request. This new programming interface enables you to access a lot more information about synchronization results and it also enables you to queue synchronizations, making them easier to manage.

See [“Dbmlsync API” on page 90](#).

- **DBTools interface for dbmlsync** You can use the `DBTools` interface for `dbmlsync` to integrate synchronization functionality into your SQL Anywhere synchronization client applications. All the SQL Anywhere database management utilities are built on `DBTools`.

See [“DBTools interface for dbmlsync” on page 304](#).

Scripted upload

You can also override the use of the client transaction log and define your own upload stream. See [“Scripted upload” on page 310](#).

Dbmlsync API

The Dbmlsync API provides a programming interface that allows MobiLink client applications written in C++ or .NET to launch synchronizations and receive feedback about the progress of the synchronizations they request. The API is intended to integrate synchronization seamlessly into your applications.

This programming interface enables you to access a lot more information about synchronization results and it also enables you to queue synchronizations, making them easier to manage.

For more information about Dbmlsync API, see:

- [“Dbmlsync .NET API reference” on page 253](#)
- [“Dbmlsync C++ API reference” on page 231](#)

SQL Anywhere client logging

When you create MobiLink applications with SQL Anywhere remote databases, there are two types of client log file that you should be aware of:

- dbmlsync message log
- SQL Anywhere transaction log

dbmlsync message log

By default, dbmlsync messages are sent to the dbmlsync message window. In addition, you can send the output to a message log file using the `-o` or `-ot` options. The following partial command line sends output to a log file named `dbmlsync.dbs`.

```
dbmlsync -o dbmlsync.dbs ...
```

Logging dbmlsync activity is particularly useful during the development process and when troubleshooting.

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

- Use the `-o` option to specify a log file and append output to it.
- Use the `-ot` option to specify a log file, but delete the contents the file before appending output to it.
- In addition to `-o` or `-ot`, use the `-os` option to specify the size at which the log file is renamed and a new file is started with the original name.

For more information, see:

- [“-o dbmlsync option” on page 111](#)
- [“-ot dbmlsync option” on page 112](#)
- [“-os dbmlsync option” on page 111](#)

When no message log file is specified, all output is displayed in the dbmlsync messages window. When a message log file is specified, less output is sent to the dbmlsync messages window.

You can control what information is logged to the message log file and displayed in the dbmlsync messages window using the `-v` option. Verbose output is not recommended for normal operation in a production environment because it can slow performance.

For more information, see “[-v dbmlsync option](#)” on page 123.

SQL Anywhere transaction log

See “[Transaction log files](#)” on page 81.

Running MobiLink on Mac OS X

You can run the MobiLink server and the SQL Anywhere MobiLink client on Mac OS X. You cannot run UltraLite on Mac OS X.

To synchronize a MobiLink consolidated database on Mac OS X, you can use the SQL Anywhere ODBC driver as the driver manager. See “[Create an ODBC data source on Mac OS X](#)” [[SQL Anywhere Server - Database Administration](#)].

To start the MobiLink server on Mac OS X

1. Start SyncConsole.

In the Finder, double-click SyncConsole. The SyncConsole application is located in `/Applications/SQLAnywhere12`.

2. Choose **File** » **New** » **MobiLink Server**.

3. Configure the MobiLink server:

- a. In the **Connection Parameters** field, enter the following string:

```
dsn=dsn-name
```

The *dsn-name* is a SQL Anywhere ODBC data source name. For information on creating ODBC data sources, see “[Creating ODBC data sources](#)” [[SQL Anywhere Server - Database Administration](#)].

If *dsn-name* has spaces, surround the string with double quotes. For example:

```
dsn="SQL Anywhere 12 Demo"
```

- b. Set options in the **Options** field, if desired.

The **Options** field allows you to control many aspects of MobiLink server behavior. For a complete list of options, see “[mlsrv12 syntax](#)” [[MobiLink - Server Administration](#)].

4. Click **Start** to start the MobiLink server.

The database server messages window appears and displays messages, showing that the server is ready to accept synchronization requests.

To start dbmlsync on Mac OS X

1. Start SyncConsole.

In the **Finder**, double-click **SyncConsole**. The SyncConsole application is located in */Applications/SQLAnywhere12*.

2. Choose **File » New » MobiLink Client**.

The client options window appears. It has many configuration options, which correspond to dbmlsync command line options. For a complete listing, see [“dbmlsync syntax” on page 93](#).

The options on the **Login**, **Database**, **Network**, and **Advanced** tabs all define the connection from the MobiLink client to the SQL Anywhere remote database. Often, you only need to specify an ODBC data source on the **Login** tab to connect.

The options on the **DBMLSync** tab define aspects of the connection to the MobiLink server. If these features are defined in a remote database publication and subscription, then you can leave the options on this tab empty.

To run the sample database on Mac OS X

1. Source the *sa_config* configuration script.

For more information, see [“Setting environment variables on Unix and Mac OS X” \[SQL Anywhere Server - Database Administration\]](#).

2. Set up an ODBC data source. For example:

```
dbdsn -w "SQL Anywhere 12 Demo"  
-c "uid=DBA;pwd=sql;dbf=/Applications/SQLAnywhere12/System/demo.db"
```

3. Run the MobiLink server. For example:

```
mlsrv12 -c "dsn=SQL Anywhere 12 Demo"
```

Version considerations

In order for dbmlsync to function properly, both the major and minor versions of dbmlsync.exe must match those of the database server. In addition, the major version of the database file must match that of dbmlsync.exe, and the minor version of the database file must be equal to or less than the minor version of *dbmlsync.exe*. The database file's version is the latest version to which it has been upgraded.

For example, the 9.0.2 version of dbmlsync should only be used with the 9.0.2 version of the database server (*dbeng9.exe*) and it can work with database files from versions 9.00, 9.01 and 9.02.

MobiLink SQL Anywhere client utility (dbmlsync)

dbmlsync syntax

Use the dbmlsync utility to synchronize SQL Anywhere remote databases with a consolidated database.

Syntax

dbmlsync [*options*] [*transaction-logs-directory*]

Option	Description
<i>@data</i>	Read in options from the specified environment variable or configuration file. See “ @data dbmlsync option ” on page 97.
-a	Do not prompt for input again on error. See “ -a dbmlsync option ” on page 97.
-ap	Specify authentication parameters. See “ -ap dbmlsync option ” on page 98.
-ba filename	Apply a download file. See “ -ba dbmlsync option ” on page 98.
-bc filename	Create a download file. See “ -bc dbmlsync option ” on page 99.
-be string	When creating a download file, add a string. See “ -be dbmlsync option ” on page 99.
-bg	When creating a download file, make it suitable for new remotes. See “ -bg dbmlsync option ” on page 99.
-bk	Enables background synchronization. See “ -bk dbmlsync option ” on page 100.
-bkr	Controls dbmlsync behavior after a background synchronization is interrupted. See “ -bkr dbmlsync option ” on page 101.
-c connection-string	Supply database connection parameters in the form <i>parm1=value1; parm2=value2,...</i> that are used to connect to the remote database. If you do not supply this option, a window appears and you must supply connection information. See “ -c dbmlsync option ” on page 101.
-ci size	Sets the initial size of the dbmlsync cache. See “ -ci dbmlsync option ” on page 102.
-cl size	Set the minimum size threshold for the dbmlsync cache file. See “ -cl dbmlsync option ” on page 102.

Option	Description
-cm <i>size</i>	Set the maximum size limit for the dbmlsync cache file. See “ -cm dbmlsync option ” on page 103.
-d	Drop any other connections to the database whose locks conflict with the articles to be synchronized. See “ -d dbmlsync option ” on page 103.
-dc	Continue a previously failed download. See “ -dc dbmlsync option ” on page 103.
-dl	Display log messages on the dbmlsync messages window. See “ -dl dbmlsync option ” on page 104.
-do	Disables scanning of offline transaction logs. See “ -do dbmlsync option ” on page 104.
-drs <i>bytes</i>	For restartable downloads, specify the maximum amount of data that may need to be resent after a communications failure. See “ -drs dbmlsync option ” on page 105.
-ds	Perform a download-only synchronization. See “ -ds dbmlsync option ” on page 106.
-e “ <i>option=value</i> ”...	Specify extended options. See “ MobiLink SQL Anywhere client extended options ” on page 126.
-eh	Ignore errors that occur in hook functions.
-ek <i>key</i>	Specify the remote database encryption key. See “ -ek dbmlsync option ” on page 107.
-ep	Prompt for the remote database encryption key. See “ -ep dbmlsync option ” on page 108.
-eu	Specify extended options for upload defined by most recent -n option. See “ -eu dbmlsync option ” on page 108.
-is	Ignore schedule. See “ -is dbmlsync option ” on page 108.
-k	Close window on completion. See “ -k dbmlsync option (deprecated) ” on page 109.
-l	List available extended options. See “ -l dbmlsync option ” on page 109.
-mn <i>password</i>	Specify new MobiLink password. See “ -mn dbmlsync option ” on page 109.

Option	Description
-mp <i>password</i>	Specify MobiLink password. See “ -mp dbmlsync option ” on page 110.
-n <i>name</i>	Specify synchronization publication name(s). See “ -n dbmlsync option (deprecated) ” on page 110.
-o <i>logfile</i>	Log output messages to this file. See “ -o dbmlsync option ” on page 111.
-os <i>size</i>	Specify a maximum size for the message log file, at which point the log is renamed. See “ -os dbmlsync option ” on page 111.
-ot <i>logfile</i>	Delete the contents of the message log file and then log output messages to it. See “ -ot dbmlsync option ” on page 112.
-p	Disable logscan polling. See “ -p dbmlsync option ” on page 112.
-pc+	Maintain an open connection to the MobiLink server between synchronizations. See “ -pc+ dbmlsync option ” on page 113.
-pd <i>dllname;...</i>	Preload specified DLLs for Windows Mobile. See “ -pd dbmlsync option ” on page 114.
-pi	Test that you can connect to MobiLink. See “ -pi dbmlsync option ” on page 114.
-po	Specifies the port on which dbmlsync listens. See “ -po dbmlsync option ” on page 115.
-pp <i>number</i>	Set logscan polling period. See “ -pp dbmlsync option ” on page 115.
-q	Run in minimized window. See “ -q dbmlsync option ” on page 116.
-qc	Shut down dbmlsync when synchronization is finished. See “ -qc dbmlsync option ” on page 116.
-r [<i>a</i> <i>b</i>]	Use client progress values for upload retry. See “ -r dbmlsync option ” on page 116.
-s <i>name</i>	Specify synchronization subscription name(s). See “ -s dbmlsync option ” on page 117.
-sc	Reload schema information before each synchronization. See “ -sc dbmlsync option ” on page 118.
-sm	Causes dbmlsync to start in server mode. See “ -sm dbmlsync option ” on page 118.

Option	Description
-sp <i>sync profile</i>	Add options from the synchronization profile to the synchronization options specified on the command line. See “-sp dbmlsync option” on page 119.
-tu	Perform transactional upload. See “-tu dbmlsync option” on page 119.
-u <i>ml_username</i>	Specify the MobiLink user to synchronize. See “-u dbmlsync option (deprecated)” on page 120.
-ud	For Unix only. Run dbmlsync as a daemon. See “-ud dbmlsync option” on page 121.
-ui	For Linux with X window, starts dbmlsync in shell mode if a usable display isn't available. See “-ui dbmlsync option” on page 122.
-uo	Perform upload-only synchronization. See “-uo dbmlsync option” on page 122.
-urc <i>row-estimate</i>	Specify an estimate of the number of rows to upload. See “-urc dbmlsync option” on page 123.
-ux	For Solaris and Linux, open the dbmlsync messages window. See “-ux dbmlsync option” on page 123.
-v [<i>levels</i>]	Verbose operation. See “-v dbmlsync option” on page 123.
-wc <i>classname</i>	Specify a window class name. See “-wc dbmlsync option” on page 124.
-x	Rename and restart the transaction log. See “-x dbmlsync option” on page 125.
<i>transaction-logs-directory</i>	Specify the location of the transaction log. See Transaction Log File, below.

Remarks

Run dbmlsync to synchronize a SQL Anywhere remote database with a consolidated database.

To locate and connect to the MobiLink server, dbmlsync uses the information on the publication, synchronization user, synchronization subscription, or the dbmlsync command line.

Transaction log file The *transaction-logs-directory* is the directory that contains the transaction log for the SQL Anywhere remote database. There is an active transaction log and zero or more transaction log archive files, all of which may be required by dbmlsync to determine what to upload. You must specify this parameter if the following are all true:

- the contents of the working log file have been truncated and the file has been renamed since you last synchronized
- you run the dbmsync utility from a directory other than the one where the renamed log files are stored

For more information, see [“Transaction log files” on page 81](#).

dbmsync event hooks There are also dbmsync client stored procedures that can help you customize the synchronization process. For more information, see [“Introduction to dbmsync hooks” on page 178](#) and [“Event hooks for SQL Anywhere clients” on page 178](#).

Using dbmsync For more information about using dbmsync, see [“Initiating synchronization” on page 79](#).

See also

- [“Initiating synchronization” on page 79](#)
- [“Event hooks for SQL Anywhere clients” on page 178](#)
- [“Dbmsync API” on page 90](#)
- [“DBTools interface for dbmsync” on page 304](#)

@data dbmsync option

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

Syntax

```
dbmsync @data ...
```

Remarks

With this option, you can put command line options in an environment variable or configuration file. If both exist with the name you specify, the environment variable is used.

For more information about configuration files, see [“Using configuration files” \[SQL Anywhere Server - Database Administration\]](#).

If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file.

See [“File Hiding utility \(dbfhide\)” \[SQL Anywhere Server - Database Administration\]](#).

-a dbmsync option

Certain types of errors (such as an incorrect MobiLink password) normally cause dbmsync to display a dialog prompting the user to correct the values. The -a option prevents dbmsync from prompting after these errors occur.

Syntax

```
dbmlsync -a ...
```

-ap dbmlsync option

Supplies parameters passed to the `authenticate_parameters` script and to authentication parameters on the MobiLink server.

Syntax

```
dbmlsync -ap "parameters,..." ...
```

Remarks

Use when you use the `authenticate_parameters` connection script or authentication parameters on the server. For example,

```
dbmlsync -ap "parm1,parm2,parm3"
```

The parameters are sent to the MobiLink server and passed to the `authenticate_parameters` script or other events on the consolidated database.

See also

- “Authentication parameters” [*MobiLink - Server Administration*]
- “authenticate_parameters connection event” [*MobiLink - Server Administration*]
- “AuthParms synchronization profile option” on page 161

-ba dbmlsync option

Applies a download file.

Syntax

```
dbmlsync -ba "filename" ...
```

Remarks

Specify the name of an existing download file to be applied to the remote database. You can optionally specify a path. If you do not specify a path, the default location is the directory where `dbmlsync` was started.

See also

- “MobiLink file-based download” [*MobiLink - Server Administration*]
- “-bc dbmlsync option” on page 99
- “-be dbmlsync option” on page 99
- “-bg dbmlsync option” on page 99
- “ApplyDnldFile synchronization profile option” on page 162

-bc dbmlsync option

Creates a download file.

Syntax

```
dbmlsync -bc "filename" ...
```

Remarks

Create a download file with the specified name. You should use the file extension *.df* for download files.

You can optionally specify a path. If you do not specify a path, the default location is the dbmlsync current working directory, which is the directory where dbmlsync was started.

Optionally, when creating a download file, you can use the -be option to specify a string that can be validated at the remote database, and the -bg option to create a download file for new remote databases.

See also

- “MobiLink file-based download” [*MobiLink - Server Administration*]
- “-ba dbmlsync option” on page 98
- “-be dbmlsync option” on page 99
- “-bg dbmlsync option” on page 99
- “CreateDnldFile synchronization profile option” on page 165

-be dbmlsync option

When creating a download file, this option specifies an extra string to be included in the file.

Syntax

```
dbmlsync -bc "filename" -be "string" ...
```

Remarks

The string can be used for authentication or other purposes. It is passed to the `sp_hook_dbmlsync_validate_download_file` stored procedure on the remote database when the download file is applied.

See also

- “sp_hook_dbmlsync_validate_download_file” on page 229
- “MobiLink file-based download” [*MobiLink - Server Administration*]
- “-bc dbmlsync option” on page 99
- “-ba dbmlsync option” on page 98
- “DnldFileExtra synchronization profile option” on page 166

-bg dbmlsync option

When creating a download file, this option creates a file that can be used by remote databases that have not yet synchronized.

Syntax

dbmlsync -bc "filename" -bg ...

Remarks

The `-bg` option causes the download file to update the generation numbers on the remote database.

This option allows you to build a download file that can be applied to remote databases that have never synchronized. Otherwise, you must perform a synchronization before you apply a download file.

Download files built with the `-bg` option should be snapshot downloads. Timestamp-based downloads do not work with remote databases that have not synchronized because the last download timestamp on a new remote is by default January 1, 1900, which is earlier than the last download timestamp in the download file. For timestamp-based file-based downloads to work, the last download timestamp in the download file must be the same or earlier than on the remote.

Do not apply `-bg` download files to remote databases that have already synchronized if your system depends on functionality provided by generation numbers as this option circumvents that functionality.

See also

- [“MobiLink file-based download” \[MobiLink - Server Administration\]](#)
- [“-ba dbmlsync option” on page 98](#)
- [“-bc dbmlsync option” on page 99](#)
- [“MobiLink generation numbers” \[MobiLink - Server Administration\]](#)
- [“Synchronizing new remotes” \[MobiLink - Server Administration\]](#)
- [“UpdateGenNum synchronization profile option” on page 175](#)

-bk dbmlsync option

Enables background synchronization.

Syntax

dbmlsync -bk "connection-string" ...

Remarks

During a background synchronization, the database engine drops the `dbmlsync` connection to the remote database and rolls back any uncommitted `dbmlsync` operations, if another connection is waiting for access to any database resource that `dbmlsync` has locked. This allows the other connections to go forward without waiting for the synchronization to complete. Depending on the operations `dbmlsync` had outstanding when its connection is dropped, there may still be a significant delay for the waiting connection as the database rolls back the `dbmlsync` uncommitted changes.

When the `dbmlsync` connection is dropped, the synchronization in progress will fail and report errors.

See also

- [“-bkr dbmlsync option” on page 101](#)
- [“MobiLink synchronization profiles” on page 157](#)
- [“Background synchronization profile option” on page 162](#)

-bkr dbmlsync option

Controls the behavior of dbmlsync after a background synchronization is interrupted.

Syntax

dbmlsync -bkr *num*...

Remarks

num is an integer greater than or equal to -1.

If *num* is -1 then dbmlsync retries an interrupted synchronization until it completes, successfully or unsuccessfully, without being interrupted. If *num* is 0 then dbmlsync does not retry the interrupted synchronization. If *num* is greater than 0 then dbmlsync retries the synchronization up to *num* times until it completes. After *num* attempts, if the synchronization has not completed, then it is run as a foreground synchronization so it will complete without interruption.

By default BackgroundRetry is 0. It is an error to set BackgroundRetry to a non-zero value when the Background option has not been set to TRUE. See [“-bk dbmlsync option” on page 100](#).

The BackgroundRetry is ignored when the dbmlsync API or the SQL SYNCHRONIZE statement is used.

See also

- [“-bk dbmlsync option” on page 100](#)
- [“BackgroundRetry synchronization profile option” on page 162](#)

-c dbmlsync option

Specifies connection parameters for the remote database.

Syntax

dbmlsync -c "*connection-string*" ...

Remarks

The connection string must give dbmlsync permission to connect to the SQL Anywhere remote database with DBA or REMOTE DBA authority. It is recommended that you use a user ID with REMOTE DBA authority.

Specify the connection string in the form *keyword=value*, with multiple parameters separated by semicolons. If any of the parameter names contain spaces, you need to enclose the connection string in double quotes.

If you do not specify `-c`, a `dbmlsync Setup` window appears. You can specify the remaining command line options in the connection window fields.

For a complete list of connection parameters for connecting to SQL Anywhere databases, see [“Connection parameters” \[SQL Anywhere Server - Database Administration\]](#).

-ci dbmlsync option

Sets the initial size of the `dbmlsync` cache.

Syntax

`dbmlsync -ci size [K | M | P]...`

Remarks

The *size* is the initial cache size, in bytes, used by `dbmlsync` to store synchronization data. You can optionally use the suffix `K` or `M` to specify units of kilobytes or megabytes, respectively.

To specify the size as a percentage of the total physical memory in the system, specify a number between 0 and 100, followed by the letter `p`. For example, `-ci 30p` sets the initial cache size to 30% of the physical memory.

See also

- [“CacheInit synchronization profile option” on page 163](#)

-cl dbmlsync option

Set the minimum size to which the `dbmlsync` cache file is reduced.

Syntax

`dbmlsync -cl size [K | M | P]...`

Remarks

The *size* is the smallest size, in bytes, that the `dbmlsync` cache can be reduced to. You can optionally use the suffix `K` or `M` to specify units of kilobytes or megabytes, respectively.

To specify the size as a percentage of the total physical memory in the system, specify a number between 0 and 100, followed by the letter `p`. For example, `-cl 5p` ensures the cache size will not drop below 5% of the physical memory.

See also

- [“CacheMin synchronization profile option” on page 164](#)

-cm dbmlsync option

Set the maximum size limit for the dbmlsync cache.

Syntax

dbmlsync -cm *size* [**K** | **M** | **P**]...

Remarks

The *size* is the largest size, in bytes, that the dbmlsync cache can grow to. You can optionally use the suffix K or M to specify units of kilobytes or megabytes, respectively.

To specify the size as a percentage of the total physical memory in the system, specify a number between 0 and 100, followed by the letter p. For example, `-cm 60p` limits the maximum size of the cache to 60% of physical memory.

See also

- [“CacheMax synchronization profile option” on page 164](#)

-d dbmlsync option

Drops conflicting locks to the remote database.

Syntax

dbmlsync -d ...

Remarks

In cases where dbmlsync must obtain locks on the tables being synchronized, if another connection has a lock on one of these tables, the synchronization may fail or be delayed. Specifying this option forces SQL Anywhere to drop any other connections to the remote database that hold conflicting locks so that synchronization can proceed immediately.

See also

- [“Concurrency during synchronization” on page 82](#)
- [“KillConnections synchronization profile option” on page 169](#)

-dc dbmlsync option

Restart a previously failed download.

Syntax

dbmlsync -dc ...

Remarks

By default, if dbmsync fails during a download it doesn't apply any of the download data to the remote database. However, it stores the part of the download it did receive in a temporary file on the remote device, so that if you specify -dc the next time you synchronize, it can more quickly complete the download. When you specify -dc, dbmsync attempts to download the part of the previous download that it did not receive. If it is able to download the remaining data, it applies the complete download to your remote database, otherwise the synchronization fails.

If there is any new data to be uploaded when you use -dc, the restartable download fails.

You can also restart a failed download using the ContinueDownload extended option or the sp_hook_dbmsync_end hook.

See also

- [“Resuming failed downloads” \[MobiLink - Server Administration\]](#)
- [“ContinueDownload \(cd\) extended option” on page 130](#)
- [“sp_hook_dbmsync_end” on page 203](#)
- [“DownloadReadSize \(drs\) extended option” on page 133](#)
- [“ContinueDownload synchronization profile option” on page 165](#)

-dl dbmsync option

Displays messages in the dbmsync messages window or command prompt, and the message log file.

Syntax

```
dbmsync -dl ...
```

Remarks

Normally when output is logged to a file, more messages are written to the log file than to the dbmsync window. This option forces dbmsync to write information normally only written to the file to the window as well. Using this option may reduce the speed of synchronization.

-do dbmsync option

Disables scanning of offline transaction logs.

Syntax

```
dbmsync -do ...
```

Remarks

If transaction log files for multiple databases are stored in a single directory, dbmsync might not be able to sync from any of these databases, even if there is no offline transaction log file for any of these databases. If the -d option is used with dbmsync, dbmsync does not attempt to scan any offline

transaction logs and should be able to synchronize from a database that is stored with all the other databases in a single directory.

If this option is used and if offline transaction logs are required, dbmlsync is not able to synchronize.

This option cannot be used with -x option.

-drs dbmlsync option

For restartable downloads, specifies the maximum number of bytes that may need to be resent after a communications failure.

Syntax

```
dbmlsync -drs bytes ...
```

Remarks

The -drs option specifies a download read size that is only useful when doing restartable downloads.

Dbmlsync reads the download in chunks. The download read size defines the size of these chunks. When a communication error occurs, dbmlsync loses the entire chunk that was being processed. Depending on when the error occurs, the number of bytes lost ranges between 0 and the download read size -1. So for example, if the DownloadReadSize is 100 bytes and an error occurs after reading 497 bytes, the last 97 bytes read are lost. Bytes that are lost in this way are resent when the download is restarted.

In general, larger download read size values result in better performance on successful synchronizations but result in more data being resent when an error occurs.

The typical use of this option is to reduce the default size when communication is unreliable.

The default is **32767**. If you set this option to a value larger than 32767, the value 32767 is used.

You can also specify the download read size using the DownloadReadSize extended option.

See also

- [“DownloadReadSize \(drs\) extended option” on page 133](#)
- [“Resuming failed downloads” \[*MobiLink - Server Administration*\]](#)
- [“ContinueDownload \(cd\) extended option” on page 130](#)
- [“sp_hook_dbmlsync_end” on page 203](#)
- [“-dc dbmlsync option” on page 103](#)

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -drs 100
```

-ds dbmlsync option

Performs a download-only synchronization.

Syntax

dbmlsync -ds ...

Remarks

When download-only synchronization occurs, dbmlsync does not upload any database changes. However, it does upload information about the schema and progress offset.

In addition, dbmlsync ensures that changes on the remote are not overwritten during download-only synchronization. It does this by scanning the log to detect rows with operations waiting to be uploaded. If any of these rows is modified by the download, the download is rolled back and the synchronization fails. If the synchronization fails for this reason, you must do a full synchronization to correct the problem.

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

When -ds is used, the ConflictRetries extended option is ignored. dbmlsync never retries a download-only synchronization. When a download-only synchronization fails, it continues to fail until a normal synchronization is performed.

For a list of the scripts that must be defined for download-only synchronization, see [“Required scripts” \[MobiLink - Server Administration\]](#).

See also

- [“Upload-only and download-only synchronizations” \[MobiLink - Server Administration\]](#)
- [“DownloadOnly \(ds\) extended option” on page 132](#)
- [“Download-only publications” on page 71](#)
- [“DownloadOnly synchronization profile option” on page 166](#)

-e dbmlsync option

Specifies extended options.

Syntax

dbmlsync -e *extended-option=value*; ...

Remarks

Extended options can be specified by their long form or short form. For a list of extended options, see [“MobiLink SQL Anywhere client extended options” on page 126](#).

Use the dbmlsync -l option to get a list of all the extended options. See [“-l dbmlsync option” on page 109](#).

Options specified on the command line with the `-e` option apply to all synchronizations requested on the command line. For example, in the following command line the extended option `drs=512` applies to the synchronization of both `sub1` and `sub2`.

```
dbmlsync -e "drs=512" -s sub1 -s sub2
```

You can review extended options in the `dbmlsync` message log and the `SYSSYNC` system view.

To specify extended options for a single upload, use the `-eu` option.

See also

- [“-eu dbmlsync option” on page 108](#)
- [“SYSSYNC system view” \[SQL Anywhere Server - SQL Reference\]](#)
- [“sp_hook_dbmlsync_set_extended_options” on page 219](#)
- [“ExtOpt synchronization profile option” on page 168](#)

Example

The following `dbmlsync` command line illustrates how you can set extended options when you start `dbmlsync`:

```
dbmlsync -e "adr=host=localhost;dir=c:\db\logs"...
```

-eh dbmlsync option

Ignores errors that occur in hook functions.

Syntax

```
dbmlsync -eh ...
```

See also

- [“IgnoreHookErrors synchronization profile option” on page 168](#)

-ek dbmlsync option

Allows you to specify the encryption key for strongly encrypted remote databases directly on the command line.

Syntax

```
dbmlsync -ek key ...
```

Remarks

If you have a strongly encrypted remote database, you must provide the encryption key to use the database or transaction log in any way, including offline transactions. For strongly encrypted databases, you must specify either `-ek` or `-ep`, but not both. The command fails if you do not specify a key for a strongly encrypted database.

-ep dbmlsync option

Prompt for the encryption key for the remote database.

Syntax

```
dbmlsync -ep ...
```

Remarks

This option causes a window to appear, in which you enter the encryption key. It provides an extra measure of security by never allowing the encryption key to be seen in clear text. For strongly encrypted remote databases, you must specify either -ek or -ep, but not both. The command fails if you do not specify a key for a strongly encrypted database.

-eu dbmlsync option

Specifies extended upload options.

Syntax

```
dbmlsync -s subscription-name -eu keyword=value;...
```

```
dbmlsync -n publication-name -eu keyword=value;...
```

Remarks

Extended options that are specified on the command line with the -eu option apply only to the synchronization specified by the -n option or the -s option they follow. For example, on the following command line, the extended option eh=on applies only to the synchronization of subscription sub2.

```
dbmlsync -s sub1 -s sub2 -eu eh=on
```

For an explanation of how extended options are processed when they are set in more than one place, see [“MobiLink SQL Anywhere client extended options” on page 126](#).

For a complete list of extended options, see [“MobiLink SQL Anywhere client extended options” on page 126](#).

-is dbmlsync option

Ignores the Schedule extended option.

Syntax

```
dbmlsync -is ...
```

Remarks

Ignore extended options that schedule synchronization.

For information about scheduling, see [“Scheduling synchronization” on page 87](#).

See also

- [“IgnoreScheduling synchronization profile option” on page 169](#)

-k dbmlsync option (deprecated)

Shuts down dbmlsync when synchronization is finished. Dbmlsync does not shut down if an error occurs during the synchronization unless the -c or -ot option is also specified.

This option is deprecated. Use -qc instead.

Syntax

dbmlsync -k ...

See also

- [“-qc dbmlsync option” on page 116](#)

-l dbmlsync option

Lists available extended options.

Syntax

dbmlsync -l ...

See also

- [“MobiLink SQL Anywhere client extended options” on page 126](#)

-mn dbmlsync option

Supplies a new password for the MobiLink user being synchronized.

Syntax

dbmlsync -mn *password* ...

Remarks

Changes the MobiLink user's password.

For more information, see [“MobiLink users” on page 4](#).

See also

- [“MobiLinkPwd \(mp\) extended option” on page 140](#)
- [“NewMobiLinkPwd \(mn\) extended option” on page 140](#)
- [“-mp dbmlsync option” on page 110](#)
- [“NewMobiLinkPwd synchronization profile option” on page 171](#)

-mp dbmlsync option

Supplies the password of the MobiLink user being synchronized.

Syntax

```
dbmlsync -mp password ...
```

Remarks

Supplies the password for MobiLink user authentication.

For more information, see [“MobiLink users” on page 4](#).

See also

- [“MobiLinkPwd \(mp\) extended option” on page 140](#)
- [“NewMobiLinkPwd \(mn\) extended option” on page 140](#)
- [“-mn dbmlsync option” on page 109](#)

-n dbmlsync option (deprecated)

Note

This option has been deprecated. It is recommended that you use the -s dbmlsync option instead. See [“-s dbmlsync option” on page 117](#).

To use the dbmlsync -s option you need to determine the subscription name for the subscription you want to synchronize. You can determine the subscription name using the following query:

```
SELECT subscription_name
FROM syssync JOIN sys.syspublication
WHERE site_name = <ml_user> AND publication_name = <pub_name>;
```

Replace <ml_user> with the MobiLink user you are synchronizing. This is the value specified by the -u option on the dbmlsync command line. See [“-u dbmlsync option \(deprecated\)” on page 120](#).

Replace <pub_name> with the name of the publication being synchronized. This is the value specified with the -n option on the dbmlsync command line. See [“-n dbmlsync option \(deprecated\)” on page 110](#).

Specifies the publication(s) to synchronize.

Syntax

dbmlsync -n *pubname* ...

Remarks

You can supply more than one -n option to synchronize more than one synchronization publication.

There are two ways to use -n to synchronize multiple publications:

- Specify **-n pub1 , pub2 , pub3** to upload pub1, pub2, and pub3 in one upload followed by one download.

In this case, if you have set extended options on the publications or subscriptions, only the options set on the first publication in the list and its subscription are used. Extended options set on subsequent publications and subscriptions are ignored.

- Specify **-n pub1 -n pub2 -n pub3** to synchronize pub1, pub2, and pub3 in three separate sequential synchronizations.

When successive synchronizations occur very quickly, such as when you specify **-n pub1 -n pub2**, dbmlsync could start processing a synchronization when the server is still processing the previous synchronization. In this case, the second synchronization fails with an error indicating that concurrent synchronizations are not allowed. If you run into this situation, you can define an `sp_hook_dbmlsync_delay` stored procedure to create a delay before each synchronization. Usually a few seconds to a minute is a enough delay.

For more information, see [“sp_hook_dbmlsync_delay” on page 191](#).

See also

- [“Publication synchronization profile option” on page 172](#)

-o dbmlsync option

Specifies the name of the dbmlsync message log file.

Syntax

dbmlsync -o *filename* ...

Remarks

Append output to a log file. Default is to send output to the screen.

See also

- [“-os dbmlsync option” on page 111](#)
- [“-ot dbmlsync option” on page 112](#)

-os dbmlsync option

Specifies a maximum size for the dbmlsync message log file, at which point the log is renamed.

Syntax

dbmlsync -os *size* [**K** | **M** | **G**]...

Remarks

The *size* is the maximum file size for the dbmlsync message logs, specified in units of bytes. Use the suffix *k*, *m* or *g* to specify units of kilobytes, megabytes or gigabytes, respectively. By default, there is no size limit. The minimum size limit is 10K.

Before the dbmlsync utility 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 dbmlsync utility renames the output file to *yymmddxx.dbs*, where *yymmdd* represents the year, month, and day, and *xx* are sequential characters ranging from AA to ZZ.

This option allows you to manually delete old log files and free up disk space.

See also

- [“-o dbmlsync option” on page 111](#)
- [“-ot dbmlsync option” on page 112](#)

-ot dbmlsync option

Deletes the contents of the specified file and then logs output messages to it.

Syntax

dbmlsync -ot *logfile* ...

Remarks

The functionality is the same as the -o option except the contents of the message log file are deleted when dbmlsync starts up, before any messages are written to it.

See also

- [“-o dbmlsync option” on page 111](#)
- [“-os dbmlsync option” on page 111](#)

-p dbmlsync option

Disables logscan polling.

Syntax

dbmlsync -p ...

Remarks

To build an upload, dbmsync must scan the transaction log. Usually it does this at the beginning of synchronization. However, when synchronizations are scheduled or when the `sp_hook_dbmsync_delay` hook is used, dbmsync by default scans the log in the pause that occurs before synchronization. This behavior is more efficient because when synchronization begins the log is already at least partially scanned. This default behavior is called logscan polling.

Logscan polling is on by default but only has an effect when synchronizations are scheduled using the Schedule extended option or when `sp_hook_dbmsync_delay` hook is used. When in effect, polling occurs at set intervals. The default interval is 1 minute, but it can be changed with the dbmsync `-pp` option.

This option is identical to the extended option `DisablePolling=on`.

See also

- [“DisablePolling \(p\) extended option” on page 131](#)
- [“PollingPeriod \(pp\) extended option” on page 142](#)
- [“-pp dbmsync option” on page 115](#)

-pc+ dbmsync option

Maintain a persistent connection to the MobiLink server between synchronizations.

Syntax

```
dbmsync -pc+ ...
```

Remarks

When this option is specified, dbmsync connects to the MobiLink server as usual, but it then keeps that connection open for use during subsequent synchronizations. A persistent connection is closed when any of the following occur:

- An error occurs that causes a synchronization to fail.
- Liveness checking has timed out.
See [“timeout” on page 50](#).
- A synchronization is initiated in which the communication type or address are different. This could mean that the settings are different (for example, a different host is specified), or that they are specified in a different way (for example, the same host and port are specified, but in a different order).

When a persistent connection is closed, a new connection is opened that is also persistent.

This option is most useful when the client synchronizes frequently and the cost of establishing a connection to the server is high.

By default, persistent connections are not maintained.

-pd dbmsync option

Preload specified DLLs for Windows Mobile.

Syntax

```
dbmsync -pd dllname;
```

Remarks

When running dbmsync on Windows Mobile, if you are using encrypted communication streams you must use the -pd option to ensure that the appropriate DLLs are loaded at startup. Otherwise, dbmsync does not attempt to load the DLLs until they are needed. Loading these DLLs late is prone to failure due to resource limitations on Windows Mobile.

The following are the DLLs that need to be loaded for each communication protocol:

Protocol	DLL
ECC	mlcecc12.dll
RSA	mlcrsa12.dll
FIPS	mlcrsafips12.dll

You should specify multiple DLLs as a semicolon-separated list. For example:

```
-pd mlcrsafips12.dll;mlcrsa12.dll
```

-pi dbmsync option

Pings a MobiLink server.

Syntax

```
dbmsync -pi -c connection_string ...
```

Remarks

When you use -pi, dbmsync connects to the remote database, retrieves information required to connect to the MobiLink server, connects to the server, and authenticates the specified MobiLink user. When the MobiLink server receives a ping, it connects to the consolidated database, authenticates the user, and then sends the authenticating user status and value back to the client. If the MobiLink user name cannot be found in the ml_user system table and the MobiLink server is running with the command line option -zu+, the MobiLink server adds the user to the ml_user MobiLink system table.

To adequately test your connection, you should use the -pi option with all the synchronization options you want to use to synchronize with dbmsync. When -pi is included, dbmsync does not perform a synchronization.

If the ping succeeds, the MobiLink server issues an information message. If the ping does not succeed, it issues an error message.

When you start dbmlsync with `-pi`, the MobiLink server can execute only the following scripts, if they exist:

- `begin_connection`
- `authenticate_user`
- `authenticate_user_hashed`
- `authenticate_parameters`
- `end_connection`

See also

- [“Ping synchronization profile option” on page 172](#)

-po dbmlsync option

When dbmlsync is in server mode, this option specifies the port on which dbmlsync listens for connections from clients.

Syntax

`dbmlsync -po port number ...`

Remarks

This option can only be used with the `-sm` option.

See also

- [“-sm dbmlsync option” on page 118](#)

-pp dbmlsync option

Specifies the frequency of log scans.

Syntax

`dbmlsync -pp number [h | m | s]...`

Remarks

To build an upload, dbmlsync must scan the transaction log. Usually it does this at the beginning of synchronization. However, when synchronizations are scheduled or when the `sp_hook_dbmlsync_delay` hook is used, dbmlsync by default scans the log in the pause that occurs before synchronization. This behavior is more efficient because when synchronization begins the log is already at least partially scanned. This default behavior is called logscan polling.

This option specifies the interval between log scans. Use the suffix s, m, h, or d to specify seconds, minutes, hours or days, respectively. The default is **1** minute. If you do not specify a suffix, the default unit of time is minutes.

See also

- [“PollingPeriod \(pp\) extended option” on page 142](#)
- [“DisablePolling \(p\) extended option” on page 131](#)
- [“-p dbmlsync option” on page 112](#)

-q dbmlsync option

Starts the MobiLink synchronization client in a minimized window.

Syntax

`dbmlsync -q ...`

-qc dbmlsync option

Shuts down dbmlsync when synchronization is finished.

Syntax

`dbmlsync -qc ...`

Remarks

When used, dbmlsync exits after synchronization is completed if the synchronization was successful or if a message log file was specified using the `-o` or `-ot` options.

See also

- [“-o dbmlsync option” on page 111](#)
- [“-ot dbmlsync option” on page 112](#)

-r dbmlsync option

Specifies that the remote offset should be used when there is disagreement between the offsets in the remote and consolidated databases.

The `-rb` option indicates that the remote offset should be used if it is less than the consolidated offset (such as when the remote database has been restored from backup). The `-r` option is provided for backward compatibility and is identical to `-rb`. The `-ra` option indicates that the remote offset should be used if it is greater than the consolidated offset. This option is provided only for very rare circumstances and may cause data loss.

Syntax

```
dbmlsync { -r | -ra | -rb } ...
```

Remarks

For information about progress offsets, see [“Progress offsets” on page 66](#).

-rb If the remote database is restored from backup, the default behavior may cause data to be lost. In this case, the first time you run dbmlsync after the remote database is restored, you should specify -rb. When you use -rb, the upload continues from the offset recorded in the remote database if the offset recorded in the remote is less than that obtained from the consolidated database. If you use -rb and the offset in the remote is not less than the offset from the consolidated database, an error is reported and the synchronization is aborted.

The -rb option may result in some data being uploaded that has already been uploaded. This can result in conflicts in the consolidated database and should be handled with appropriate conflict resolution scripts.

-ra The -ra option should be used only in very rare cases. If you use -ra, the upload is retried starting from the offset obtained from the remote database if the remote offset is greater than the offset obtained from the consolidated database. If you use -ra and the offset in the remote is not greater than the offset from the consolidated database, an error is reported and the synchronization is aborted.

The -ra option should be used with care. If the offset mismatch is the result of a restore of the consolidated database, changes that happened in the remote database in the gap between the two offsets are lost. The -ra option may be useful when the consolidated database has been restored from backup and the remote database transaction log has been truncated at the same point as the remote offset. In this case, all data that was uploaded from the remote database is lost from the point of the consolidated offset to the point of the remote offset.

See also

- [“RemoteProgressGreater synchronization profile option” on page 173](#)
- [“RemoteProgressLess synchronization profile option” on page 174](#)

-s dbmlsync option

Specifies the subscription(s) to be synchronized.

Syntax

```
dbmlsync -s subname ...
```

Remarks

This option replaces the -n dbmlsync option.

There are two ways to use -s to synchronize multiple subscriptions:

- Specify `-s sub1 , sub2 , sub3` to synchronize sub1, sub2, and sub3 in one upload followed by one download.

In this case, if you have set extended options on the subscriptions, only the options set on the first subscription in the list are used. Extended options set on subsequent subscriptions ignored.

- Specify `-s sub1 -s sub2 -s sub3` to synchronize sub1, sub2, and sub3 in three separate sequential synchronizations, each with its own upload and download.

When successive synchronizations occur very quickly, such as when you specify `-s sub1 -s sub2`, dbmsync could start processing a synchronization when the server is still processing the previous synchronization. In this case, the second synchronization fails with an error indicating that concurrent synchronizations are not allowed. If you run into this situation, you can define an `sp_hook_dbmsync_delay` stored procedure to create a delay before each synchronization. Usually a few seconds to a minute is a enough delay.

For more information, see [“sp_hook_dbmsync_delay” on page 191](#).

See also

- [“Subscription synchronization profile option” on page 174](#)

-sc dbmsync option

Specifies that dbmsync should reload schema information before each synchronization.

Syntax

```
dbmsync -sc ...
```

Remarks

Before version 9.0, dbmsync reloaded schema information from the database before each synchronization. The information that was reloaded includes foreign key relationships, publication definitions, extended options stored in the database, and information about database settings. Loading this information is time-consuming and often the information does not change between synchronizations.

Starting with version 9.0, by default dbmsync loads schema information only at startup. Specify `-sc` if you want the information to be loaded before every synchronization.

-sm dbmsync option

Causes dbmsync to start in server mode.

Syntax

```
dbmsync -sm ...
```

Remarks

When in server mode, dbmsync starts up and waits for connections from applications using the Dbmsync API or the SQL `SYNCHRONIZE` statement.

This option should only be used when starting a dbmsync server from the command line.

Normally dbmlsync servers are started directly using the Dbmlsync API or the SQL SYNCHRONIZE statement. This option **should not** be used when either of those methods is used.

See also

- [“-po dbmlsync option” on page 115](#)

-sp dbmlsync option

When -sp is used, the options in the specified synchronization profile are added to those specified on the command line for the synchronization.

Syntax

dbmlsync -sp *sync profile ...*

Remarks

If equivalent options are specified on the command line and in the synchronization profile, then the options on the command line override those specified in the profile.

See also

- [“CREATE SYNCHRONIZATION PROFILE statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“ALTER SYNCHRONIZATION PROFILE statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“DROP SYNCHRONIZATION PROFILE statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“MobiLink synchronization profiles” on page 157](#)

-tu dbmlsync option

Specifies that each transaction on the remote database should be uploaded as a separate transaction within one synchronization.

Syntax

dbmlsync -tu ...

Remarks

When you use -tu, you create a **transactional upload**: dbmlsync uploads each transaction on the remote database as a distinct transaction. The MobiLink server applies and commits each transaction separately when it is received.

When you use -tu, the order of transactions on the remote database is always preserved on the consolidated database. However, the order of operations in a transaction may not be preserved, for two reasons:

- MobiLink always applies updates based on foreign key relationships. For example, when data is changed in child and parent tables, MobiLink inserts data into the parent table before the child table, but deletes data from the child before the parent. If your remote operations do not follow this order, the order of operations differ on the consolidated database.
- Operations within a transaction are coalesced. This means that if you change the same row three times in one transaction, only the final form of the row is uploaded.

If a transactional upload is interrupted, the data that was not sent is sent in the next synchronization. Typically, only the transactions that were not successfully completed are sent at that time. Sometimes, such as when the upload failure occurs during the first synchronization of a subscription, dbmlsync resends all transactions.

When you do not use `-tu`, MobiLink coalesces all changes on the remote database into one transaction in the upload. This means that if you change the same row three times between synchronizations, regardless of the number of remote transactions, only the final form of the row is uploaded. This default behavior is efficient and is optimal in many situations.

However, in certain situations you may want to preserve remote transactions on the consolidated database. For example, you may want to define triggers on the consolidated database that act on transactions as they occur in the remote database.

In addition, there are advantages to breaking up the upload into smaller transactions. Many consolidated databases are optimized for small transactions, so sending a very large transaction is not efficient or may cause too much contention. Also, when you use `-tu` you may not lose the entire upload if there are communications errors during the upload. When you use `-tu` and there is an upload error, all successfully uploaded transactions are applied.

The `-tu` option makes MobiLink behave in a manner that is very close to SQL Remote. The main difference is that SQL Remote replicates all changes to the remote database in the order they occur, without coalescing. To mimic this behavior, you must commit after each database operation on the remote database.

You cannot use `-tu` with the Increment extended option or with scripted uploads.

See also

- [“-tx mlsrv12 option”](#) [*MobiLink - Server Administration*]
- [“Uploading data from self-referencing tables”](#) [*MobiLink - Server Administration*]
- [“TransactionalUpload synchronization profile option”](#) on page 175

-u dbmlsync option (deprecated)

Note

This option has been deprecated. It is recommended that you use the `-s dbmlsync` option instead. See [“-s dbmlsync option” on page 117](#).

To use the `dbmlsync -s` option you need to determine the subscription name for the subscription you want to synchronize. You can determine the subscription name using the following query:

```
SELECT subscription_name
FROM syssync JOIN sys.syspublication
WHERE site_name = <ml_user> AND publication_name = <pub_name>;
```

Replace `<ml_user>` with the MobiLink user you are synchronizing. This is the value specified by the `-u` option on the `dbmlsync` command line. See [“-u dbmlsync option \(deprecated\)” on page 120](#).

Replace `<pub_name>` with the name of the publication being synchronized. This is the value specified with the `-n` option on the `dbmlsync` command line. See [“-n dbmlsync option \(deprecated\)” on page 110](#).

Specifies the MobiLink user name.

Syntax

```
dbmlsync -u ml_username ...
```

Remarks

You can specify only one user on the `dbmlsync` command line, where `ml_username` is the name used in the `FOR` clause of the `CREATE SYNCHRONIZATION SUBSCRIPTION` statement corresponding to the subscription to be processed.

This option should be used in conjunction with `-n publication` to identify the subscription on which `dbmlsync` should operate. Each subscription is uniquely identified by an `ml_username, publication` pair.

You can only specify one user name on the command line. All subscriptions to be synchronized in a single run must involve the same user. The `-u` option can be omitted if each publication that is specified on the command line with the `-n` option has only one subscription.

See also

- [“MLUser synchronization profile option \(deprecated\)” on page 171](#)

-ud dbmlsync option

For Unix platforms only, instructs `dbmlsync` to run as a daemon.

Syntax

```
dbmlsync -ud ...
```

Remarks

Unix platforms only.

If you run dbmlsync as a daemon, you should also supply either the `-o` or `-ot` option to log output information.

When you start dbmlsync as a daemon, its permissions are controlled by the current user's umask setting. It is recommended that you set the umask value before starting dbmlsync to ensure that dbmlsync has the appropriate permissions.

See also

- [“-o dbmlsync option” on page 111](#)
- [“-ot dbmlsync option” on page 112](#)

-ui dbmlsync option

For Linux with X Windows server support, starts dbmlsync in shell mode if a usable display is not available.

Syntax

```
dbmlsync -ui ...
```

Remarks

When this option is used, dbmlsync tries to start with X Windows. If this fails, it starts in shell mode.

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

-uo dbmlsync option

Specifies that synchronization only includes an upload.

Syntax

```
dbmlsync -uo...
```

Remarks

During upload-only synchronization, dbmlsync prepares and sends an upload to MobiLink exactly as it would in a normal full synchronization. However, instead of sending a download back down, MobiLink sends only an acknowledgement indicating if the upload was successfully committed.

For a list of the scripts that must be defined for upload-only synchronization, see [“Required scripts” \[MobiLink - Server Administration\]](#).

See also

- [“Upload-only and download-only synchronizations” \[MobiLink - Server Administration\]](#)
- [“DownloadOnly \(ds\) extended option” on page 132](#)
- [“UploadOnly \(uo\) extended option” on page 150](#)
- [“UploadOnly synchronization profile option” on page 176](#)

-urc dbmlsync option

Specifies an estimate of the number of rows to be uploaded in a synchronization.

Syntax

```
dbmlsync -urc row-estimate ...
```

Remarks

To improve performance, you can specify an estimate of the number of rows to upload in a synchronization. This setting is especially useful when you are uploading a large number of rows. A higher estimate results in faster uploads but more memory usage.

Synchronization proceeds correctly regardless of the specified estimate.

See also

- [“For large uploads, estimate the number of rows” \[MobiLink - Server Administration\]](#)
- [“UploadRowCnt synchronization profile option” on page 176](#)

-ux dbmlsync option

On Linux, opens a dbmlsync messages window where messages are displayed.

Syntax

```
dbmlsync -ux...
```

Remarks

When -ux is specified, dbmlsync 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, dbmlsync fails to start.

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

On Windows, the dbmlsync messages window appears automatically.

See also

- [“-q dbmlsync option” on page 116](#)

-v dbmlsync option

Allows you to specify what information is logged to the message log file and displayed in the synchronization window. A high level of verbosity may affect performance and should normally be used in the development phase only.

Syntax

dbmlsync -v [*levels*] ...

Remarks

The -v options affect the message log file and synchronization window. You only have a message log if you specify -o or -ot on the dbmlsync command line.

If you specify -v alone, a small amount of information is logged but more information than if -v is omitted.

The values of *levels* are as follows. You can use one or more of these options at once; for example, -vnrsu or -v+cp.

- **+** Turn on all logging options except for c and p.
- **c** Expose the connect string in the log.
- **p** Expose the MobiLink password in the log.
- **n** Log the number of rows that were uploaded and downloaded.
- **o** Log information about the command line options and extended options that you have specified.
- **r** Log the values of rows that were uploaded and downloaded.
- **s** Log messages related to hook scripts.
- **u** Log information about the upload.

There are extended options that have similar functionality to the -v options. If you specify both -v and the extended options and there are conflicts, the -v option overrides the extended option. If there is no conflict, the verbosity logging options are additive—all options that you specify are used. When logging verbosity is set by extended option, the logging does not take effect immediately, so startup information is not logged. By the time of the first synchronization, the logging behavior is identical between the -v options and the extended options.

See also

- [“Verbose \(v\) extended option” on page 150](#)
- [“VerboseHooks \(vs\) extended option” on page 151](#)
- [“VerboseMin \(vm\) extended option” on page 152](#)
- [“VerboseOptions \(vo\) extended option” on page 153](#)
- [“VerboseRowCounts \(vn\) extended option” on page 154](#)
- [“VerboseRowValues \(vr\) extended option” on page 155](#)
- [“-o dbmlsync option” on page 111](#)
- [“-ot dbmlsync option” on page 112](#)
- [“Verbosity synchronization profile option” on page 177](#)

-wc dbmlsync option

Specifies a window class name.

Syntax

```
dbmlsync -wc class-name ...
```

Remarks

This option specifies a window class name that can be used to wake up dbmlsync whenever it is in hover mode, such as when scheduling is enabled or when you are using server-initiated synchronization.

In addition, the window class name identifies the application for Microsoft ActiveSync synchronization. The class name must be given when registering the application for use with Microsoft ActiveSync synchronization.

This option applies only to Windows.

See also

- [“Registering SQL Anywhere clients for Microsoft ActiveSync” on page 86](#)
- [“Using Microsoft ActiveSync synchronization” on page 83](#)
- INFINITE keyword in [“Schedule \(sch\) extended option” on page 143](#)
- [“Scheduling synchronization” on page 87](#)

Example

```
dbmlsync -wc dbmlsync_$message_end...
```

-x dbmlsync option

Renames and restarts the transaction log.

Syntax

```
dbmlsync -x [ size [ K | M | G ] ...
```

Remarks

When the optional size is specified the log is renamed if it is larger than that size in bytes. Use the suffix k, m or g to specify units of kilobytes, megabytes or gigabytes, respectively. The default size is 0.

If backups are not routinely performed at the remote database, the transaction log continues to grow. As an alternative to using the -x option to control transaction log size, you can use a SQL Anywhere event handler to control the size of the transaction log.

See also

- [“Automating tasks using schedules and events” \[SQL Anywhere Server - Database Administration\]](#)
- [“delete_old_logs option \[SQL Remote\]” \[SQL Anywhere Server - Database Administration\]](#)
- [“CREATE EVENT statement” \[SQL Anywhere Server - SQL Reference\]](#)

MobiLink SQL Anywhere client extended options

Introduction to dbmsync extended options

Extended options can be specified on the dbmsync command line using the -e or -eu options, or they can be stored in the database. You store extended options in the database by using Sybase Central, by using the sp_hook_dbmsync_set_extended_options event hook, or by using the OPTION clause in any of the following statements:

- CREATE SYNCHRONIZATION SUBSCRIPTION
- ALTER SYNCHRONIZATION SUBSCRIPTION
- CREATE SYNCHRONIZATION USER
- ALTER SYNCHRONIZATION USER
- CREATE SYNCHRONIZATION SUBSCRIPTION without specifying a synchronization user (which associates extended options with a publication)

Priority order

Dbmsync combines options stored in the database with those specified on the command line. If conflicting options are specified, dbmsync resolves them as follows. In the following list, options specified by methods occurring earlier in the list take precedence over those occurring later in the list.

1. Options specified in the sp_hook_dbmsync_set_extended_options event hook.
2. Options specified in the command line that aren't extended options. (For example, -ds overrides -e "ds=off".)
3. Options specified in the command line with the -eu option.
4. Options specified in the command line with the -e option.
5. Options specified for the subscription, whether by a SQL statement or in Sybase Central. When you use the **Deploy Synchronization Model Wizard** to deploy a MobiLink model, extended options are set for you and are specified in the subscription.
6. Options specified for the MobiLink user, whether by a SQL statement or in Sybase Central.
7. Options specified for the publication, whether by a SQL statement or in Sybase Central.

Note

This priority order also affects connection parameters, such as those specified with the TYPE and ADDRESS options in the SQL statements mentioned above.

You can review extended options in the log and the SYSSYNC system view.

For information about how extended options can be used to tune synchronization, see [“Using dbmlsync extended options” on page 80](#).

See also

- [“-e dbmlsync option” on page 106](#)
- [“-eu dbmlsync option” on page 108](#)
- [“CREATE SYNCHRONIZATION SUBSCRIPTION statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“ALTER SYNCHRONIZATION SUBSCRIPTION statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“CREATE SYNCHRONIZATION USER statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“ALTER SYNCHRONIZATION USER statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“CREATE PUBLICATION statement \[MobiLink\] \[SQL Remote\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“ALTER PUBLICATION statement \[MobiLink\] \[SQL Remote\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“SYSSYNC system view” \[SQL Anywhere Server - SQL Reference\]](#)
- [“sp_hook_dbmlsync_set_extended_options” on page 219](#)

Example

The following dbmlsync command line illustrates how you can set extended options when you start dbmlsync:

```
dbmlsync -e "adr=host=localhost;dir=c:\db\logs"...
```

The following SQL statement illustrates how you can store extended options in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION TO mypub
  FOR mluser
  ADDRESS 'host=localhost'
  OPTION schedule='weekday@11:30am-12:30pm', dir='c:\db\logs'
```

The following dbmlsync command line opens the usage screen that lists options and their syntax:

```
dbmlsync -l
```

BufferDownload (bd) extended option

Specifies whether the entire download from the MobiLink server should be read into the cache before applying it to the remote database.

Syntax

```
bd={ON | OFF}; ...
```

```
BufferDownload={ON | OFF}; ...
```

Remarks

The default is ON. The default results in lower load on the MobiLink server and using it should improve server side throughput.

When BufferDownload is set to off, dbmlsync applies the download as it is read.

CommunicationAddress (adr) extended option

Specifies network protocol options for connecting to the MobiLink server.

Syntax

`adr=protocol-option; ...`

`CommunicationAddress=protocol-option; ...`

Remarks

For parameters, see [“MobiLink client network protocol option summary” on page 23](#).

You must ensure that all subscriptions for a MobiLink user are synchronized to only one consolidated database. Otherwise, you may experience data loss and unpredictable behavior.

Use the CommunicationType extended option to specify the type of network protocol.

See [“CommunicationType \(ctp\) extended option” on page 129](#).

See also

- [“MobiLink client network protocol options” on page 23](#)
- [“Relay Server configuration file” \[Relay Server\]](#)

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "adr=host=localhost"
```

To specify multiple network protocol options on the command line, enclose them in single quotes. For example,

```
dbmlsync -e "adr='host=somehost;port=5001'"
```

To store the Address or CommunicationType in the database, you can use an extended option or you can use the ADDRESS clause. For example,

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO sales_publication  
  FOR ml_user1  
  ADDRESS 'host=localhost;port=2439'
```

CommunicationType (ctp) extended option

Specifies the type of network protocol to use for connecting to the MobiLink server.

Syntax

ctp=*network-protocol*; ...

CommunicationType=*network-protocol*; ...

Remarks

network-protocol can be one of **tcPIP**, **tls**, **http**, or **https**. The default is **tcPIP**.

You must ensure that all subscriptions for a MobiLink user are synchronized to only one consolidated database. Otherwise, you may experience data loss and unpredictable behavior.

See also

- “Encrypting MobiLink client/server communications” [[SQL Anywhere Server - Database Administration](#)]
- “CommunicationAddress (adr) extended option” on page 128

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "ctp=https"
```

The following SQL illustrates how to store this option in the database.

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO sales_publication  
  FOR ml_user1  
  OPTION ctp='tcPIP'
```

ConflictRetries (cr) extended option

Specifies the number of retries if the download fails because of conflicts.

Syntax

cr=*number*, ...

ConflictRetries=*number*, ...

Remarks

When tables are not locked during synchronization, it is possible for operations to be applied to the database between the time the upload is built and the time that the download is applied. If these changes affect rows that are also changed by the download, dbmlsync considers this to be a conflict and does not apply the download stream. When this occurs dbmlsync retries the entire synchronization. Normally the synchronization succeeds on the next attempt but it is possible for a new conflict to force a new retry. This option controls the maximum number of retries that are performed.

This option is useful only if the LockTables option is OFF, which is the default.

The default is **-1** (retries should continue indefinitely).

See also

- “Handling conflicts” [*MobiLink - Server Administration*]

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "cr=5"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
TO sales_publication  
FOR ml_user1  
OPTION cr='5';
```

ContinueDownload (cd) extended option

Restarts a previously failed download.

Syntax

cd={ ON | OFF }; ...

ContinueDownload={ ON | OFF }; ...

Remarks

If dbmlsync does not receive the entire download from the server, dbmlsync does not apply any of the download data to the remote database. However, it stores the part of the download it did receive in a temporary file on the remote device, so that it can be restarted later. When you set the extended option **cd=on**, dbmlsync restarts the download and attempts to download the part of the previous download that it did not receive. If it is able to download the remaining data, it applies the complete download to your remote database.

If there is any new data to be uploaded when you set **cd=on**, the synchronization fails without restarting the download. If the download cannot be restarted, synchronization fails.

You can also restart a download with the **-dc** option or with the **sp_hook_dbmlsync_end** hook.

See also

- “Resuming failed downloads” [*MobiLink - Server Administration*]
- “sp_hook_dbmlsync_set_extended_options” on page 219
- “-dc dbmlsync option” on page 103
- “sp_hook_dbmlsync_end” on page 203

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "cd=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO sales_publication  
  FOR ml_user1  
  OPTION cd='on';
```

DisablePolling (p) extended option

Disables automatic logscan polling.

Syntax

```
p={ON | OFF}; ...
```

```
DisablePolling={ON | OFF}; ...
```

Remarks

To build an upload, dbmlsync must scan the transaction log. Usually it does this just before synchronization. However, when synchronizations are scheduled or the `sp_hook_dbmlsync_delay` hook is used, dbmlsync by default scans the log in the time between synchronizations. This behavior is more efficient because the log is already at least partially scanned when synchronization begins. This default behavior is called logscan polling.

Logscan polling is on by default but only has an effect when synchronizations are scheduled or when `sp_hook_dbmlsync_delay` hook is used. When in effect, polling occurs at set intervals: dbmlsync scans to the end of the log, waits for the polling period, and then scans any new transactions in the log. By default, the polling period is 1 minute, but it can be changed with the `dbmlsync -pp` option or the `PollingPeriod` extended option.

The default is to not disable logscan polling (**OFF**).

This option is identical to **dbmlsync -p**.

See also

- [“PollingPeriod \(pp\) extended option” on page 142](#)
- [“-p dbmlsync option” on page 112](#)
- [“-pp dbmlsync option” on page 115](#)

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "p=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION p='on';
```

DownloadOnly (ds) extended option

Specifies that synchronization should be download-only.

Syntax

```
ds={ ON | OFF }; ...
```

```
DownloadOnly={ ON | OFF }; ...
```

Remarks

When download-only synchronization occurs, dbmlsync does not upload any row operations or data. However, it does upload information about the schema and progress offset.

In addition, dbmlsync ensures that changes on the remote that have not been uploaded are not overwritten during download-only synchronization. It does this by scanning the log to detect rows with operations waiting to be uploaded. If any of these rows is modified by the download, the download is rolled back and the synchronization fails. If the synchronization fails for this reason, you must do a full synchronization to correct the problem.

When you have remotes that are synchronized by download-only synchronization, you should regularly do a full synchronization to reduce the amount of log that is scanned by the download-only synchronization. Otherwise, the download-only synchronizations take an increasingly long time to complete. If this is a problem, you can alternatively use a download-only publication to avoid log issues during synchronization.

For a list of the scripts that must be defined for download-only synchronization, see [“Required scripts” \[MobiLink - Server Administration\]](#).

The default is **OFF** (perform both upload and download).

See also

- [“-ds dbmlsync option” on page 106](#)
- [“Download-only publications” on page 71](#)
- [“Upload-only and download-only synchronizations” \[MobiLink - Server Administration\]](#)

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "ds=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
```

```
FOR ml_user1  
OPTION ds='ON';
```

DownloadReadSize (drs) extended option

For restartable downloads, specifies the maximum amount of data that may need to be resent after a communications failure.

Syntax

```
drs=number[ K ]; ...
```

```
DownloadReadSize=number[ K ]; ...
```

Remarks

The DownloadReadSize option is only useful when doing restartable downloads.

The download read size is specified in units of bytes. Use the suffix k to optionally specify units of kilobytes.

Dbmsync reads the download in chunks. The DownloadReadSize defines the size of these chunks. When a communication error occurs, dbmsync loses the entire chunk that was being processed. Depending on when the error occurs, the number of bytes lost ranges between 0 and the DownloadReadSize -1. So for example, if the DownloadReadSize is 100 bytes and an error occurs after reading 497 bytes, the last 97 bytes read are lost. Bytes that are lost in this way are resent when the download is restarted.

In general, larger DownloadReadSize values result in better performance on successful synchronizations but result in more data being resent when an error occurs.

The typical use of this option is to reduce the default size when communication is unreliable.

The default is **32767**. If you set this option to a value larger than 32767, the value 32767 is used.

See also

- “-drs dbmsync option” on page 105
- “Resuming failed downloads” [*MobiLink - Server Administration*]
- “ContinueDownload (cd) extended option” on page 130
- “sp_hook_dbmsync_end” on page 203
- “-dc dbmsync option” on page 103

Example

The following dbmsync command line illustrates how you can set this option when you start dbmsync:

```
dbmsync -e "drs=100"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
TO sales_publication  
FOR ml_user1  
OPTION drs='100';
```

ErrorLogSendLimit (el) extended option

Specifies how much of the remote message log file dbmlsync should send to the server when synchronization error occurs.

Syntax

`el=number[K | M]; ...`

`ErrorLogSendLimit=number[K | M]; ...`

Remarks

This option is specified in units of bytes. Use the suffix k or m to specify units of kilobytes or megabytes, respectively.

This option specifies the number of bytes of the message log that dbmlsync sends to the MobiLink server when errors occur during synchronization. Set this option to **0** if you don't want any dbmlsync message log to be sent.

The default is **32K**.

When this option is non-zero, the error log is uploaded when a client-side error occurs. Not all client-side errors cause the log to be sent: the log is not sent for communication errors or errors that occur when dbmlsync is not connected to the MobiLink server. If the error occurs after the upload is sent, the error log is uploaded only if the SendDownloadAck extended option is set to ON.

If ErrorLogSendLimit is set to be large enough, dbmlsync sends the entire message log from the current session to the MobiLink server. For example, if the message log messages were appended to an old message log file, dbmlsync only sends the new messages generated in the current session. If the total length of new messages is greater than ErrorLogSendLimit, dbmlsync only uploads the messages log up to the specified size.

Note: The size of the message log is influenced by your verbosity settings. You can adjust these using the dbmlsync -v option, or by using dbmlsync extended options starting with "verbose". For more information, see [“-v dbmlsync option” on page 123](#) and the -e verbose options:

- [“Verbose \(v\) extended option” on page 150](#)
- [“VerboseHooks \(vs\) extended option” on page 151](#)
- [“VerboseMin \(vm\) extended option” on page 152](#)
- [“VerboseOptions \(vo\) extended option” on page 153](#)
- [“VerboseRowCounts \(vn\) extended option” on page 154](#)
- [“VerboseRowValues \(vr\) extended option” on page 155](#)
- [“VerboseUpload \(vu\) extended option” on page 156](#)

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "el=32k"
```

The following SQL statement illustrates how you can store this option in the database:


```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION el='32k';
```

FireTriggers (ft) extended option

Specifies that triggers should be fired on the remote database when the download is applied.

Syntax

```
ft={ ON | OFF }; ...
```

```
FireTriggers={ ON | OFF }; ...
```

Remarks

The default is **ON**.

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "ft=off"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION ft='off';
```

HoverRescanThreshold (hrt) extended option

When you are using scheduling, this limits the amount of discarded memory that is allowed to accumulate before a rescan is performed.

Syntax

```
hrt=number[ K | M ]; ...
```

```
HoverRescanThreshold=number[ K | M ]; ...
```

Remarks

Specifies memory in units of bytes. Use the suffix k or m to specify units of kilobytes or megabytes, respectively. The default is **1m**.

When more than one -n option or -s option is specified in the command line, dbmlsync may experience fragmentation which results in discarded memory. The discarded memory can only be recovered by rescanning the database transaction log. This option lets you specify a limit on the amount of discarded memory that is allowed to accumulate before the log is rescanned and the memory recovered. Another

way to control the recovery of discarded memory is to implement the `sp_hook_dbmsync_log_rescan` stored procedure.

See also

- [“sp_hook_dbmsync_log_rescan” on page 205](#)

Example

The following `dbmsync` command line illustrates how you can set this option when you start `dbmsync`:

```
dbmsync -e "hrt=2m"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO sales_publication  
  FOR ml_user1  
  OPTION hrt='2m';
```

IgnoreHookErrors (eh) extended option

Specifies that errors that occur in hook functions should be ignored.

Syntax

```
eh={ ON | OFF }; ...
```

```
IgnoreHookErrors={ ON | OFF }; ...
```

Remarks

The default is **OFF**.

This option is equivalent to the `dbmsync -eh` option.

Example

The following `dbmsync` command line illustrates how you can set this option when you start `dbmsync`:

```
dbmsync -e "eh=off"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO sales_publication  
  FOR ml_user1  
  OPTION eh='off';
```

IgnoreScheduling (isc) extended option

Specifies that the Schedule extended option should be ignored.

Syntax

`isc={ ON | OFF }; ...`

`IgnoreScheduling={ ON | OFF }; ...`

Remarks

If set to ON, dbmlsync ignores the Schedule extended option and synchronizes immediately. The default is OFF.

This option is equivalent to the dbmlsync -is option.

See also

- [“Scheduling synchronization” on page 87](#)
- [“Schedule \(sch\) extended option” on page 143](#)

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "isc=off"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
TO sales_publication
FOR ml_user1
OPTION isc='off';
```

Increment (inc) extended option

Enables incremental uploads and controls the size of upload increments.

Syntax

`inc=number[K | M]; ...`

`Increment=number[K | M]; ...`

Remarks

The value of this option specifies, very approximately, the size of each upload part in bytes. Use the suffix k or m to specify units of kilobytes or megabytes, respectively.

When this option is set to a non-zero value, uploads are sent to MobiLink in one or more parts. This could be useful if dbmlsync has difficulty maintaining a connection to the MobiLink server for long enough to complete the full upload. The default is 0.

The value of the option controls the size of each upload part as follows. Dbmlsync builds the upload by scanning the database transaction log. When this option is set, dbmlsync scans the number of bytes that are set in the option, and then continues scanning to the first point at which there are no outstanding partial transactions—the next point at which all transactions have either been committed or rolled back. It then sends what it has scanned as an upload part and resumes scanning the log from where it left off.

You cannot use the Increment extended option with scripted upload or transactional upload.

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "inc=32000"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO sales_publication  
  FOR ml_user1  
  OPTION inc='32k';
```

LockTables (lt) extended option

Specifies that tables in the publications being synchronized should be locked before synchronizing.

Syntax

lt={ ON | OFF | SHARE | EXCLUSIVE }; ...

LockTables={ ON | OFF | SHARE | EXCLUSIVE }; ...

Remarks

SHARE means that dbmlsync locks all synchronization tables in shared mode. EXCLUSIVE means that dbmlsync locks all synchronization tables in exclusive mode. For all platforms except Windows Mobile, ON is the same as SHARE. For Windows Mobile devices, ON is the same as EXCLUSIVE.

The default is OFF. This means that by default, dbmlsync does not lock any synchronization tables except for the following situations:

- If there is a publication that uses script-based upload in the current synchronization or if there is an `sp_hook_dbmlsync_schema_upgrade` hook defined in the remote database, dbmlsync locks the synchronization tables with SHARE.

Set to ON to prevent modifications during synchronization.

For more information about shared and exclusive locks, see [“How locking works” \[SQL Anywhere Server - SQL Usage\]](#) and [“LOCK TABLE statement” \[SQL Anywhere Server - SQL Reference\]](#).

For more information about locking tables in MobiLink applications, see [“Concurrency during synchronization” on page 82](#).

When synchronization tables are locked in exclusive mode (the default for Windows Mobile devices), no other connections can access the tables, and so dbmlsync stored procedures that execute on a separate connection are not able to execute if they require access to any of the synchronization tables.

For information about hooks that execute on separate connections, see [“Event hooks for SQL Anywhere clients” on page 178](#).

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "lt=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO sales_publication  
  FOR ml_user1  
  OPTION lt='on';
```

MirrorLogDirectory (mld) extended option

Specifies the location of old transaction log mirror files so that they can be deleted.

Syntax

```
mld=dir, ...
```

```
MirrorLogDirectory=dir, ...
```

Remarks

This option makes it possible for dbmlsync to delete old transaction log mirror files when either of the following two circumstances occur:

- the offline transaction log mirror is located in a different directory from the transaction log mirror
or
- dbmlsync is run on a different computer from the remote database server

In a normal setup, the active transaction log mirror and renamed transaction log mirror files are located in the same directory, and dbmlsync is run on the same computer as the remote database, so this option is not required and old transaction log mirror files are automatically deleted.

Transaction logs in this directory are only affected if the delete_old_logs database option is set to On, Delay, or *n* days.

See also

- [“delete_old_logs option \[SQL Remote\]” \[SQL Anywhere Server - Database Administration\]](#)

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "mld=c:\tmp\file"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO sales_publication
```

```
FOR ml_user1  
OPTION mld='c:\tmp\file';
```

MobiLinkPwd (mp) extended option

Specifies the MobiLink password.

Syntax

```
mp=password; ...
```

```
MobiLinkPwd=password; ...
```

Remarks

Specifies the password used to connect to the MobiLink server. This password should be the correct password for the MobiLink user whose subscriptions are being synchronized. The default is null.

If the MobiLink user already has a password, use the extended option **-e mn** to change it.

See also

- [“NewMobiLinkPwd \(mn\) extended option” on page 140](#)
- [“-mn dbmlsync option” on page 109](#)
- [“-mp dbmlsync option” on page 110](#)

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "mp=password"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
TO sales_publication  
FOR ml_user1  
OPTION mp='password';
```

NewMobiLinkPwd (mn) extended option

Specifies a new MobiLink password.

Syntax

```
mn=new-password; ...
```

```
NewMobiLinkPwd=new-password; ...
```

Remarks

Specifies a new password for the MobiLink user whose subscriptions are being synchronized. Use this option when you want to change an existing password. The default is not to change the password.

See also

- [“MobiLinkPwd \(mp\) extended option” on page 140](#)
- [“-mn dbmlsync option” on page 109](#)
- [“-mp dbmlsync option” on page 110](#)

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "mp=oldpassword;mn=newpassword"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO sales_publication  
  FOR ml_user1  
  OPTION mp='oldpassword';mn='newpassword'
```

NoSyncOnStartup (nss) extended option

Prevents dbmlsync from synchronizing on startup when a scheduling option would otherwise cause that to happen.

Syntax

```
nss={ on | off }; ...
```

```
NoSyncOnStartup={ on | off }; ...
```

Remarks

This option has an effect only when the schedule extended option is used with the EVERY or INFINITE clause. These scheduling options cause dbmlsync to automatically synchronize on startup.

The default is off.

When you set NoSyncOnStartup to on and use a schedule with the INFINITE clause, a synchronization does not occur until a window message is received.

When you set NoSyncOnStartup to on and use a schedule with the EVERY clause, the first synchronization after startup occurs after the amount of time specified in the EVERY clause.

This setting does not affect the behavior of the schedule in any way other than at dbmlsync startup.

See also

- [“Schedule \(sch\) extended option” on page 143](#)
- [“Scheduling synchronization” on page 87](#)

Example

The following partial dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "schedule=EVERY:01:00;nss=off"...
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO sales_publication  
  FOR ml_user1  
  OPTION nss='off', schedule='EVERY:01:00';
```

OfflineDirectory (dir) extended option

Specifies the path containing offline transaction logs.

Syntax

```
dir=path; ...
```

```
OfflineDirectory=path; ...
```

Remarks

By default, dbmlsync checks for renamed logs in the same directory as the online transaction log. This option only needs to be specified if the renamed offline transaction logs are located in a different directory.

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "dir=c:\db\logs"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO sales_publication  
  FOR ml_user1  
  OPTION dir='c:\db\logs';
```

PollingPeriod (pp) extended option

Specifies the logscan polling period.

Syntax

```
pp=number[S | M | H | D]; ...
```

```
PollingPeriod=number[S | M | H | D]; ...
```

Remarks

This option specifies the interval between log scans. Use the suffix s, m, h, or d to specify seconds, minutes, hours or days, respectively. The default is **1** minute. If you do not specify a suffix, the default unit of time is minutes.

Logscan polling occurs only when you are scheduling synchronizations or using the `sp_hook_dbmsync_delay` hook.

For an explanation of logscan polling, see [“DisablePolling \(p\) extended option” on page 131](#).

This option is identical to **dbmsync -pp**.

See also

- [“DisablePolling \(p\) extended option” on page 131](#)
- [“-pp dbmsync option” on page 115](#)
- [“-p dbmsync option” on page 112](#)
- [“sp_hook_dbmsync_delay” on page 191](#)
- [“Schedule \(sch\) extended option” on page 143](#)

Example

The following dbmsync command line illustrates how you can set this option when you start dbmsync:

```
dbmsync -e "pp=5"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION pp='5';
```

Schedule (sch) extended option

Specifies a schedule for synchronization.

Syntax

sch=*schedule*; ...

Schedule=*schedule*; ...

schedule : { **EVERY**:*hhhh:mm* | **INFINITE** | *singleSchedule* }

hhhh : **00** ... **9999**

mm : **00** ... **59**

singleSchedule : *day* @*hh:mm* [**AM** | **PM**] [-*hh:mm* [**AM** | **PM**]] ,...

hh : **00** ... **24**

mm : **00** ... **59**

day :

EVERYDAY | **WEEKDAY** | **MON** | **TUE** | **WED** | **THU** | **FRI** | **SAT** | **SUN** | *dayOfMonth*

dayOfMonth : **0**... **31**

Remarks

EVERY The EVERY keyword causes synchronization to occur on startup, and then repeat indefinitely after the specified time period. If the synchronization process takes longer than the specified period, synchronization starts again immediately.

To avoid having a synchronization occur when dbmlsync starts, use the extended option `NoSyncOnStartup`. See [“NoSyncOnStartup \(nss\) extended option” on page 141](#).

singleSchedule Given one or more single schedules, synchronization occurs only at the specified days and times.

An interval is specified as `@hh:mm-hh:mm` (with optional specification of AM or PM). If AM or PM is not specified, a 24-hour clock is assumed. 24:00 is interpreted as 00:00 on the next day. When an interval is specified, synchronization occurs, starting at a random time within the interval. The interval provides a window of time for synchronization so that multiple remote databases with the same schedule do not cause congestion at the MobiLink server by synchronizing at exactly the same time.

The interval end time is always interpreted as following the start time. When the time interval includes midnight, it ends on the next day. If dbmlsync is started midway through the interval, synchronization occurs at a random time before the end time.

EVERYDAY EVERYDAY is all seven days of the week.

WEEKDAY WEEKDAY is Monday through Friday.

Days of the week are Mon, Tue, and so on. You may also use the full forms of the day, such as Monday. You must use the full forms of the day names if the language you are using is not English, is not the language requested by the client in the connection string, and is not the language which appears in the server messages window.

dayOfMonth To specify the last day of the month regardless of the length of the month, set the *dayOfMonth* to 0.

INFINITE The INFINITE keyword causes dbmlsync to synchronize on startup, and then not to synchronize again until synchronization is initiated by another program sending a window message to dbmlsync. You can use the dbmlsync extended option `NoSyncOnStartup` to avoid the initial synchronization. For more information, see [“NoSyncOnStartup \(nss\) extended option” on page 141](#).

You can use this option in conjunction with the dbmlsync `-wc` option to wake up dbmlsync and perform a synchronization. For more information, see [“-wc dbmlsync option” on page 124](#).

If a previous synchronization is still incomplete at a scheduled time, the scheduled synchronization commences when the previous synchronization completes.

The default is no schedule.

The Schedule option is ignored when the Dbmlsync API is used or when the SQL SYNCHRONIZE statement is used.

The IgnoreScheduling extended option and the -is option instruct dbmsync to ignore scheduling, so that synchronization is immediate. For more information, see [“IgnoreScheduling \(isc\) extended option” on page 136](#).

For more information about scheduling, see [“Scheduling synchronization” on page 87](#).

Example

The following dbmsync command line illustrates how you can set this option when you start dbmsync:

```
dbmsync -e "sch=WEEKDAY@8:00am,SUN@9:00pm"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION sch='WEEKDAY@8:00am,SUN@9:00pm';
```

ScriptVersion (sv) extended option

Specifies a script version.

Caution

It is strongly recommended that you specify the script version using the SCRIPT VERSION clause on the CREATE SYNCHRONIZATION SUBSCRIPTION and ALTER SYNCHRONIZATION SUBSCRIPTION statements instead of using the ScriptVersion extended option because using the SCRIPT VERSION clause greatly simplifies the problem of doing schema upgrades.

The ScriptVersion (sv) extended option overrides the value stored using the SCRIPT VERSION clause and should only be used for backward compatibility purposes or for the rare case where you need to explicitly specify the script version used for a synchronization.

See [“CREATE SYNCHRONIZATION SUBSCRIPTION statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#) and [“ALTER SYNCHRONIZATION SUBSCRIPTION statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#).

Syntax

```
sv=version-name; ...
```

```
ScriptVersion=version-name; ...
```

Remarks

The script version determines which scripts are run by MobiLink on the consolidated database during synchronization. The default script version name is **default**.

Example

The following dbmsync command line illustrates how you can set this option when you start dbmsync:

```
dbmlsync -e "sv=SysAd001"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO sales_publication  
  FOR ml_user1  
  OPTION sv='SysAd001';
```

SendColumnNames (scn) extended option

Specifies that column names should be sent in the upload for use by direct row handling and by mlreplay.

Syntax

```
scn={ ON | OFF }; ...
```

```
SendColumnNames={ ON | OFF }; ...
```

Remarks

The default is **ON**.

Because column names are sent by default, the `ml_add_column` stored procedure is not required for most deployments. If you want `ml_add_column` names to be used, this option must be set to **OFF**. This approach is required if you want to refer to remote column names using aliases, or if the column information added using `ml_add_column` is preferred over the information automatically sent up from the remote. See “[ml_add_column system procedure](#)” [*MobiLink - Server Administration*].

The column names are used by the MobiLink server for direct row handling. When using the row handling API to refer to columns by name rather than by index, this option should be set to **ON**. See “[Direct row handling](#)” [*MobiLink - Server Administration*].

If you are using the `mlreplay` utility, this option should be set to **ON** so the replay API has meaningful column names.

Example

The following `dbmlsync` command line illustrates how you can set this option when you start `dbmlsync`:

```
dbmlsync -e "scn=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO sales_publication  
  FOR ml_user1  
  OPTION scn='on';
```

SendDownloadAck (sa) extended option

Specifies that a download acknowledgement should be sent from the client to the server.

Syntax

sa={ ON | OFF }; ...

SendDownloadAck={ ON | OFF }; ...

Remarks

A download acknowledgement lets MobiLink server know for sure that a download has been applied to a remote database. You can write synchronization scripts in your consolidated database to handle the acknowledgement and perform business logic at the time of the acknowledgement. A download acknowledgement may not be sent if the network session is dropped after the client applies the download, so your scripts should allow for this possibility. See [“nonblocking_download_ack connection event”](#) [*MobiLink - Server Administration*].

Note: When SendDownloadAck is set to ON and you are in verbose mode, an acknowledgement line is written to the client log.

The default is **OFF**.

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "sa=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO sales_publication  
  FOR ml_user1  
  OPTION sa='on';
```

SendTriggers (st) extended option

Specifies that trigger actions should be sent on upload.

Syntax

st={ ON | OFF }; ...

SendTriggers={ ON | OFF }; ...

Remarks

Cascaded deletes are also considered trigger actions.

The default is **OFF**.

If two subscriptions both contain one or more of the same tables, then both subscriptions must be synchronized using the same setting for the SendTriggers option.

Note

Trigger actions that occur as a result of database changes made as part of the download phase of synchronization are never synchronized, regardless of the value of the SendTriggers option.

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "st=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO sales_publication  
  FOR ml_user1  
  OPTION st='on';
```

TableOrder (tor) extended option

Specifies the order of tables in the upload.

Syntax

tor=*tables*; ...

TableOrder=*tables*; ...

tables = *table-name* [*,table-name*], ...

Remarks

This option allows you to specify the order in which tables are uploaded. You must specify all tables that are to be uploaded. If you include tables that are not included in the synchronization, they are ignored.

The table order that you specify must ensure referential integrity. This means that if Table1 has a foreign key reference to Table2, then Table2 must be uploaded before Table1. If you do not specify tables in the appropriate order, an error occurs, except in the two following cases:

- You set TableOrderChecking=OFF.
- Your tables have a cyclical foreign key relationship. (In this case, there is no order that satisfies the rule and so the tables involved in the cycle can be uploaded in any order.)

If you do not specify TableOrder, then dbmlsync chooses an order that satisfies referential integrity.

The order of tables on the download is the same as the upload. Control of the upload table order may make writing server side scripts simpler, especially if the remote and consolidated databases have different foreign key constraints.

See also

- [“TableOrderChecking \(toc\) extended option” on page 149](#)
- [“How the upload is processed” \[MobiLink - Getting Started\]](#)
- [“Referential integrity and synchronization” \[MobiLink - Getting Started\]](#)

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "tor=admin,parent,child"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO sales_publication  
  FOR ml_user1  
  OPTION tor='admin,parent,child';
```

TableOrderChecking (toc) extended option

Lets you disable referential integrity checking on the table order specified by the TableOrder extended option.

Syntax

```
toc={ OFF | ON }; ...
```

```
TableOrderChecking={ OFF | ON }; ...
```

Remarks

In most applications, tables on the remote and consolidated databases have the same foreign key relationships. In these cases, you should leave TableOrderChecking at its default value of ON, and dbmlsync ensures that no table is uploaded before another table on which it has a foreign key. This ensures referential integrity.

This option is useful when the consolidated and remote databases have different foreign key relationships. Use it with the TableOrder extended option to specify an order of tables that satisfies the referential integrity constraints on the server even though it may violate those present on the remote.

This option is only useful when the TableOrder extended option is specified.

The default is **ON**.

See also

- [“TableOrder \(tor\) extended option” on page 148](#)
- [“How the upload is processed” \[MobiLink - Getting Started\]](#)
- [“Referential integrity and synchronization” \[MobiLink - Getting Started\]](#)

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "toc=OFF"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO sales_publication  
  FOR ml_user1  
  OPTION toc='Off';
```

UploadOnly (uo) extended option

Specifies that synchronization should only include an upload.

Syntax

```
uo={ ON | OFF }; ...
```

```
UploadOnly={ ON | OFF }; ...
```

Remarks

During upload-only synchronization, dbmlsync prepares and sends an upload to the MobiLink server exactly as in a normal full synchronization. However, instead of sending a download back down, the MobiLink server sends only an acknowledgement indicating if the upload was successfully committed.

For a list of the scripts that must be defined for upload-only synchronization, see [“Required scripts” \[MobiLink - Server Administration\]](#).

The default is **OFF**.

See also

- “Upload-only and download-only synchronizations” [\[MobiLink - Server Administration\]](#)
- “DownloadOnly (ds) extended option” on page 132

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "uo=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO sales_publication  
  FOR ml_user1  
  OPTION uo='on';
```

Verbose (v) extended option

Specifies full verbosity.

Syntax

```
v={ ON | OFF }; ...
```

```
Verbose={ ON | OFF }; ...
```

Remarks

This option specifies a high level of verbosity, which may affect performance and should normally be used in the development phase only.

This option is identical to **dbmlsync -v+**. If you specify both **-v** and the extended options and there are conflicts, the **-v** option overrides the extended option. If there is no conflict, the verbosity logging options are additive—all options that you specify are used. When logging verbosity is set by extended option, the logging does not take effect immediately, so startup information is not logged. By the time of the first synchronization, the logging behavior is identical between the **-v** options and the extended options.

For more information, see [“-v dbmlsync option” on page 123](#).

The default is **OFF**.

See also

- [“VerboseHooks \(vs\) extended option” on page 151](#)
- [“VerboseMin \(vm\) extended option” on page 152](#)
- [“VerboseOptions \(vo\) extended option” on page 153](#)
- [“VerboseRowCounts \(vn\) extended option” on page 154](#)
- [“VerboseRowValues \(vr\) extended option” on page 155](#)
- [“VerboseUpload \(vu\) extended option” on page 156](#)

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "v=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
  TO sales_publication
  FOR ml_user1
  OPTION v='on';
```

VerboseHooks (vs) extended option

Specifies that messages related to hook scripts should be logged.

Syntax

```
vs={ ON | OFF }; ...
```

```
VerboseHooks={ ON | OFF }; ...
```

Remarks

This option is identical to **dbmsync -vs**. If you specify both **-v** and the extended options and there are conflicts, the **-v** option overrides the extended option. If there is no conflict, the verbosity logging options are additive—all options that you specify are used. When logging verbosity is set by extended option, the logging does not take effect immediately, so startup information is not logged. By the time of the first synchronization, the logging behavior is identical between the **-v** options and the extended options.

For more information, see [“-v dbmsync option” on page 123](#).

The default is **OFF**.

See also

- [“Event hooks for SQL Anywhere clients” on page 178](#)
- [“Verbose \(v\) extended option” on page 150](#)
- [“VerboseMin \(vm\) extended option” on page 152](#)
- [“VerboseOptions \(vo\) extended option” on page 153](#)
- [“VerboseRowCounts \(vn\) extended option” on page 154](#)
- [“VerboseRowValues \(vr\) extended option” on page 155](#)
- [“VerboseUpload \(vu\) extended option” on page 156](#)

Example

The following **dbmsync** command line illustrates how you can set this option when you start **dbmsync**:

```
dbmsync -e "vs=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO sales_publication  
  FOR ml_user1  
  OPTION vs='on';
```

VerboseMin (vm) extended option

Specifies that a small amount of information should be logged.

Syntax

```
vm={ ON | OFF }; ...
```

```
VerboseMin={ ON | OFF }; ...
```

Remarks

This option is identical to **dbmsync -v**. If you specify both **-v** and the extended options and there are conflicts, the **-v** option overrides the extended option. If there is no conflict, the verbosity logging options are additive—all options that you specify are used. When logging verbosity is set by extended option, the logging does not take effect immediately, so startup information is not logged. By the time of the first synchronization, the logging behavior is identical between the **-v** options and the extended options.

For more information, see [“-v dbmsync option” on page 123](#).

The default is **OFF**.

See also

- [“Verbose \(v\) extended option” on page 150](#)
- [“VerboseOptions \(vo\) extended option” on page 153](#)
- [“VerboseRowCounts \(vn\) extended option” on page 154](#)
- [“VerboseRowValues \(vr\) extended option” on page 155](#)
- [“VerboseUpload \(vu\) extended option” on page 156](#)

Example

The following dbmsync command line illustrates how you can set this option when you start dbmsync:

```
dbmsync -e "vm=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO sales_publication  
  FOR ml_user1  
  OPTION vm='on';
```

VerboseOptions (vo) extended option

Specifies that information should be logged about the command line options (including extended options) that you have specified.

Syntax

```
vo={ ON | OFF }; ...
```

```
VerboseOptions={ ON | OFF }; ...
```

Remarks

This option is identical to **dbmsync -vo**. If you specify both **-v** and the extended options and there are conflicts, the **-v** option overrides the extended option. If there is no conflict, the verbosity logging options are additive—all options that you specify are used. When logging verbosity is set by extended option, the logging does not take effect immediately, so startup information is not logged. By the time of the first synchronization, the logging behavior is identical between the **-v** options and the extended options.

For more information, see [“-v dbmsync option” on page 123](#).

The default is **OFF**.

See also

- “Verbose (v) extended option” on page 150
- “VerboseMin (vm) extended option” on page 152
- “VerboseRowCounts (vn) extended option” on page 154
- “VerboseRowValues (vr) extended option” on page 155
- “VerboseUpload (vu) extended option” on page 156

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "vo=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
TO sales_publication  
FOR ml_user1  
OPTION vo='on';
```

VerboseRowCounts (vn) extended option

Specifies that the number of rows that are uploaded and downloaded should be logged.

Syntax

```
vn={ ON | OFF }; ...
```

```
VerboseRowCounts={ ON | OFF }; ...
```

Remarks

This option is identical to **dbmlsync -vn**. If you specify both **-v** and the extended options and there are conflicts, the **-v** option overrides the extended option. If there is no conflict, the verbosity logging options are additive—all options that you specify are used. When logging verbosity is set by extended option, the logging does not take effect immediately, so startup information is not logged. By the time of the first synchronization, the logging behavior is identical between the **-v** options and the extended options.

For more information, see [“-v dbmlsync option” on page 123](#).

The default is **OFF**.

See also

- “Verbose (v) extended option” on page 150
- “Verbose (v) extended option” on page 150
- “VerboseMin (vm) extended option” on page 152
- “VerboseOptions (vo) extended option” on page 153
- “VerboseRowValues (vr) extended option” on page 155
- “VerboseUpload (vu) extended option” on page 156

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "vn=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO sales_publication  
  FOR ml_user1  
  OPTION vn='on';
```

VerboseRowValues (vr) extended option

Specifies that the values of rows that are uploaded and downloaded should be logged.

Syntax

```
vr={ ON | OFF }; ...
```

```
VerboseRowValues={ ON | OFF }; ...
```

Remarks

This option is identical to **dbmlsync -vr**. If you specify both **-v** and the extended options and there are conflicts, the **-v** option overrides the extended option. If there is no conflict, the verbosity logging options are additive—all options that you specify are used. When logging verbosity is set by extended option, the logging does not take effect immediately, so startup information is not logged. By the time of the first synchronization, the logging behavior is identical between the **-v** options and the extended options.

For more information, see [“-v dbmlsync option” on page 123](#).

The default is **OFF**.

See also

- [“Verbose \(v\) extended option” on page 150](#)
- [“Verbose \(v\) extended option” on page 150](#)
- [“VerboseMin \(vm\) extended option” on page 152](#)
- [“VerboseOptions \(vo\) extended option” on page 153](#)
- [“VerboseRowCounts \(vn\) extended option” on page 154](#)
- [“VerboseUpload \(vu\) extended option” on page 156](#)

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "vr=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
  TO sales_publication
```

```
FOR ml_user1  
OPTION vr='on';
```

VerboseUpload (vu) extended option

Specifies that information about the upload stream should be logged.

Syntax

```
vu={ ON | OFF }; ...
```

```
VerboseUpload={ ON | OFF }; ...
```

Remarks

This option is identical to **dbmlsync -vu**. If you specify both **-v** and the extended options and there are conflicts, the **-v** option overrides the extended option. If there is no conflict, the verbosity logging options are additive—all options that you specify are used. When logging verbosity is set by extended option, the logging does not take effect immediately, so startup information is not logged. By the time of the first synchronization, the logging behavior is identical between the **-v** options and the extended options.

For more information, see [“-v dbmlsync option” on page 123](#).

The default is **OFF**.

See also

- [“Verbose \(v\) extended option” on page 150](#)
- [“Verbose \(v\) extended option” on page 150](#)
- [“VerboseMin \(vm\) extended option” on page 152](#)
- [“VerboseOptions \(vo\) extended option” on page 153](#)
- [“VerboseRowCounts \(vn\) extended option” on page 154](#)
- [“VerboseRowValues \(vr\) extended option” on page 155](#)

Example

The following **dbmlsync** command line illustrates how you can set this option when you start **dbmlsync**:

```
dbmlsync -e "vu=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION  
TO sales_publication  
FOR ml_user1  
OPTION vu='on';
```

MobiLink SQL statements

MobiLink statements

The following are the SQL statements used for configuring and running MobiLink SQL Anywhere clients:

- “ALTER PUBLICATION statement [MobiLink] [SQL Remote]” [[SQL Anywhere Server - SQL Reference](#)]
- “ALTER SYNCHRONIZATION PROFILE statement [MobiLink]” [[SQL Anywhere Server - SQL Reference](#)]
- “ALTER SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]” [[SQL Anywhere Server - SQL Reference](#)]
- “ALTER SYNCHRONIZATION USER statement [MobiLink]” [[SQL Anywhere Server - SQL Reference](#)]
- “CREATE PUBLICATION statement [MobiLink] [SQL Remote]” [[SQL Anywhere Server - SQL Reference](#)]
- “CREATE SYNCHRONIZATION PROFILE statement [MobiLink]” [[SQL Anywhere Server - SQL Reference](#)]
- “CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]” [[SQL Anywhere Server - SQL Reference](#)]
- “CREATE SYNCHRONIZATION USER statement [MobiLink]” [[SQL Anywhere Server - SQL Reference](#)]
- “DROP PUBLICATION statement [MobiLink] [SQL Remote]” [[SQL Anywhere Server - SQL Reference](#)]
- “DROP SYNCHRONIZATION PROFILE statement [MobiLink]” [[SQL Anywhere Server - SQL Reference](#)]
- “DROP SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]” [[SQL Anywhere Server - SQL Reference](#)]
- “DROP SYNCHRONIZATION USER statement [MobiLink]” [[SQL Anywhere Server - SQL Reference](#)]
- “GRANT REMOTE DBA statement [MobiLink] [SQL Remote]” [[SQL Anywhere Server - SQL Reference](#)]
- “START SYNCHRONIZATION DELETE statement [MobiLink]” [[SQL Anywhere Server - SQL Reference](#)]
- “START SYNCHRONIZATION SCHEMA CHANGE statement [MobiLink]” [[SQL Anywhere Server - SQL Reference](#)]
- “STOP SYNCHRONIZATION DELETE statement [MobiLink]” [[SQL Anywhere Server - SQL Reference](#)]
- “STOP SYNCHRONIZATION SCHEMA CHANGE statement [MobiLink]” [[SQL Anywhere Server - SQL Reference](#)]
- “SYNCHRONIZE statement [MobiLink]” [[SQL Anywhere Server - SQL Reference](#)]

UltraLite clients

See “UltraLite SQL statements” [[UltraLite - Database Management and Reference](#)].

MobiLink synchronization profiles

Synchronization profiles allow you to place some dbmlsync options in the database. The synchronization profile you create can contain a variety of synchronizations options.

Synchronization profiles can be created, altered and dropped using the following statements:

- [“CREATE SYNCHRONIZATION PROFILE statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“ALTER SYNCHRONIZATION PROFILE statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#)
- [“DROP SYNCHRONIZATION PROFILE statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#)

Synchronization profiles can be accessed using the `dbmsync -sp` option, the Sync method in the Dbmsync API, the SQL SYNCHRONIZE statement and the remote tasks created from central administration of remote databases. In all cases there is the ability to specify additional options that are merged with the options in the synchronization profiles. If any of the extra option specified conflicts with an option specified in the synchronization profile, the value specified by the extra options is used.

When using the `dbmsync -sp` options, all the other option on the command line are treated as extra options. The other interfaces provide a specific parameter or clause for specifying extra options.

For information about using synchronization profiles in UltraLite, see [“Synchronization profile options” \[UltraLite - Database Management and Reference\]](#).

The following options can be specified in a synchronization profile:

Long option name	Short name	Allowed values	Description
AuthParms	ap	String	Supplies parameters to the <code>authenticate_parameters</code> script and to authentication parameters. See “AuthParms synchronization profile option” on page 161 .
ApplyDnldFile	ba	String	Applies a download file. See “ApplyDnldFile synchronization profile option” on page 162 .
Background	bk	String	Enables background synchronization when set to TRUE. See “Background synchronization profile option” on page 162 .
BackgroundRetry	bkr	Integer	Controls how <code>dbmsync</code> behaves after a background synchronization is interrupted. See “BackgroundRetry synchronization profile option” on page 162 .
CacheInit	ci	Integer	Sets the initial size of the <code>dbmsync</code> cache. See “CacheInit synchronization profile option” on page 163 .
CacheMax	cm	Integer	Sets the maximum size threshold for the <code>dbmsync</code> cache. See “CacheMax synchronization profile option” on page 164 .

Long option name	Short name	Allowed values	Description
CacheMin	cl	Integer	Sets the minimum size to which the dbmsync cache can be reduced. See “CacheMin synchronization profile option” on page 164.
ContinueDownload	dc	Boolean	Restarts a previously failed download. See “ContinueDownload synchronization profile option” on page 165.
CreateDnldFile	bc	String	Creates a download file. See “CreateDnldFile synchronization profile option” on page 165.
DnldFileExtra	be	String	When creating a download file, this option specifies an extra string to be included in the file. See “DnldFileExtra synchronization profile option” on page 166.
DownloadOnly	ds	Boolean	Performs a download-only synchronization. See “DownloadOnly synchronization profile option” on page 166.
DownloadReadSize	drs	Integer	For restartable downloads, specifies the maximum amount of data that may need to be resent after a communications failure. See “DownloadReadSize synchronization profile option” on page 167.
ExtOpt	e	String	Specifies extended options. See “ExtOpt synchronization profile option” on page 168.
IgnoreHookErrors	eh	Boolean	Ignores errors that occur in hook functions. See “IgnoreHookErrors synchronization profile option” on page 168.
IgnoreScheduling	is	Boolean	Ignores scheduling instructions so that synchronization is immediate. See “IgnoreScheduling synchronization profile option” on page 169.
KillConnections	d	Boolean	Drops conflicting locks to the remote database. See “KillConnections synchronization profile option” on page 169.
LogRenameSize	x	An integer optionally followed by K or M.	Renames and restarts the transaction log after it has been scanned for upload data. See “LogRenameSize synchronization profile option” on page 170.
MobiLinkPwd	mp	String	Supplies the password of the MobiLink user. See “MobiLinkPwd synchronization profile option” on page 170.

Long option name	Short name	Allowed values	Description
MLUser	u	String	Specifies the MobiLink user name. See “MLUser synchronization profile option (deprecated)” on page 171.
NewMobiLinkPwd	mn	String	Supplies a new password for the MobiLink user. Use this option when you want to change an existing password. See “NewMobiLinkPwd synchronization profile option” on page 171.
Ping	pi	Boolean	Pings a MobiLink server to confirm communications between the client and MobiLink. See “Ping synchronization profile option” on page 172.
Publication	n	String	This option is deprecated. Specifies the publication(s) to synchronize. Note that publication can only be specified once in a synchronization profile but the command line option can be specified multiple times. See “Publication synchronization profile option” on page 172.
RemoteProgressGreater	ra	Boolean	Specifies that the remote offset should be used if it is greater than the consolidated offset. This is equivalent to the -ra option. See “RemoteProgressGreater synchronization profile option” on page 173.
RemoteProgressLess	rb	Boolean	Specifies that the remote offset should be used if it is less than the consolidated offset (such as when the remote database has been restored from backup). This is equivalent to the -rb option. See “RemoteProgressLess synchronization profile option” on page 174.
Subscription	s	String	Specifies the subscription(s) to synchronize. Note that subscription can only be specified once in a synchronization profile. See “Subscription synchronization profile option” on page 174.
TransactionalUpload	tu	Boolean	Specifies that each transaction on the remote database should be uploaded as a separate transaction within one synchronization. See “TransactionalUpload synchronization profile option” on page 175.
UpdateGenNum	bg	Boolean	When creating a download file, this option creates a file that can be used with remote databases that have not yet synchronized. See “UpdateGenNum synchronization profile option” on page 175.

Long option name	Short name	Allowed values	Description
UploadOnly	uo	Boolean	Specifies that synchronization only includes an upload, and that no downloads occur. See “UploadOnly synchronization profile option” on page 176.
UploadRowCnt	urc	Integer	Specifies an estimate of the number of rows to be uploaded in a synchronization. See “UploadRowCnt synchronization profile option” on page 176.
Verbosity		String (a comma separated list of options)	<p>Controls dbmlsync verbosity. Similar to the “Verbosity synchronization profile option” on page 177.</p> <p>The value must be a comma separated list of one or more of the following options, each of which corresponds to an existing -v option as described below:</p> <ul style="list-style-type: none"> ● BASIC - equivalent to -v ● HIGH - equivalent to -v+ ● CONNECT_STR - equivalent to -vc ● ROW_CNT - equivalent to -vn ● OPTIONS - equivalent to -vo ● ML_PASSWORD - equivalent to -vp ● ROW_DATA - equivalent to -vr

AuthParms synchronization profile option

Supplies parameters to the authenticate_parameters script and to authentication parameters.

Syntax

ap=parameters

Authparms=parameters

Remarks

Use when you use the authenticate_parameters connection script or authentication parameters.

The parameters are sent to the MobiLink server and passed to the authenticate_parameters script or other events on the consolidated database.

See also

- [“-ap dbmlsync option”](#) on page 98

Example

AuthParms=p1,p2,p3

ApplyDnldFile synchronization profile option

Applies a download file.

Syntax

ba=*filename*

ApplyDnldFile=*filename*

Remarks

Specify the name of an existing download file to be applied to the remote database.

See also

- [“-ba dbmsync option” on page 98](#)

Example

```
ApplyDnldFile=filename
```

Background synchronization profile option

Enables background synchronization when set to ON.

Syntax

bk={ON|OFF}

Background={ON|OFF}

Remarks

During a background synchronization, the database engine drops the dbmsync connection to the remote database and rolls back any uncommitted dbmsync operations if another connection is waiting for access to any database resource that dbmsync has locked. This allows the other connections to go forward without waiting for the synchronization to complete. Depending on the operations dbmsync had outstanding when its connection is dropped, there may still be a significant delay for the waiting connection as the database rolls back the dbmsync uncommitted changes.

See also

- [“-bk dbmsync option” on page 100](#)

Example

The following example shows how to use the Background synchronization profile option:

```
Background=on
```

BackgroundRetry synchronization profile option

Integer. Controls how dbmsync behaves after a background synchronization is interrupted.

Syntax

bkr=*integer*

BackgroundRetry=*integer*

Remarks

This option has a short form and long form: you can use **bkr** or **BackgroundRetry**.

Set this value as an integer greater than or equal to -1. If the value is -1 then **dbmlsync** retries an interrupted synchronization until it completes, successfully or unsuccessfully, without being interrupted. If the value is 0 then **dbmlsync** does not retry the interrupted synchronization. If the value is greater than 0 then **dbmlsync** retries the synchronization up to the number of times specified until it completes. After the specified number of attempts, if the synchronization has not completed, then it is run as a foreground synchronization so it will complete without interruption.

By default **BackgroundRetry** is 0.

It is an error to set **BackgroundRetry** to a non-zero value when the **Background** option has not been set to **ON**.

This option is ignored when synchronization is initiated using the **Dbmlsync** API or the SQL **SYNCHRONIZE** statement.

See also

- [“-bkr dbmlsync option” on page 101](#)

Example

The following example shows how to use the **BackgroundRetry** synchronization profile option:

```
BackgroundRetry=4
```

Cachelnit synchronization profile option

Sets the initial size of the **dbmlsync** cache.

Syntax

ci=size[**K** | **M** | **P**]

Cachelnit=size[**K** | **M** | **P**]

Remarks

Sets the initial size of the **dbmlsync** cache. You can optionally use the suffix **K** or **M** to specify units of kilobytes or megabytes, respectively.

To specify the size as a percentage of the total physical memory in the system, specify a number between 0 and 100, followed by the letter **p**.

See also

- [“-ci dbmsync option” on page 102](#)

Example

```
CacheInit=10k
```

CacheMin synchronization profile option

Sets the minimum size to which the dbmsync cache can be reduced.

Syntax

```
cl=size[ K | M | P ]
```

```
CacheMin=size[ K | M | P ]
```

Remarks

Set the smallest size, in bytes, that the dbmsync cache can be reduced to. You can optionally use the suffix K or M to specify units of kilobytes or megabytes, respectively.

To specify the size as a percentage of the total physical memory in the system, specify a number between 0 and 100, followed by the letter p.

See also

- [“-cl dbmsync option” on page 102](#)

Example

```
CacheMin=10k
```

CacheMax synchronization profile option

Sets the maximum size threshold for the dbmsync cache.

Syntax

```
cm=size[ K | M | P ]
```

```
CacheMax=size[ K | M | P ]
```

Remarks

Sets the largest size, in bytes, that the dbmsync cache can grow to. You can optionally use the suffix K or M to specify units of kilobytes or megabytes, respectively.

To specify the size as a percentage of the total physical memory in the system, specify a number between 0 and 100, followed by the letter p.

See also

- [“-cm dbmlsync option” on page 103](#)

Example

```
CacheMax=1m
```

ContinueDownload synchronization profile option

Restarts a previously failed download.

Syntax

```
dc={ON|OFF}
```

```
ContinueDownload={ON|OFF}
```

Remarks

By default, if dbmlsync fails during a download it does not apply any of the download data to the remote database. However, it stores the part of the download it did receive in a temporary file on the remote device, so that if you specify ContinueDownload=on the next time you synchronize, it can more quickly complete the download.

When you specify ContinueDownload=on, dbmlsync attempts to download the part of the previous download that it did not receive. If it is able to download the remaining data, it applies the complete download to your remote database, otherwise synchronization fails.

If there is any new data to be uploaded when you set ContinueDownload=on, the restartable download fails. You can also restart a failed download using the ContinueDownload extended option or the sp_hook_dbmlsync_end hook.

See also

- [“-dc dbmlsync option” on page 103](#)
- [“sp_hook_dbmlsync_end” on page 203](#)
- [“ContinueDownload \(cd\) extended option” on page 130](#)

Example

The following example shows how to use the ContinueDownload synchronization profile option:

```
ContinueDownload=on
```

CreateDnldFile synchronization profile option

Creates a download file with the specified name.

Syntax

```
bc=filename
```

CreateDnldFile=*filename*

Remarks

You should use the file extension *.df* for download files.

You can optionally specify a path. If you do not specify a path, the default location is the dbmlsync current working directory, which is the directory where dbmlsync was started.

Optionally, when creating a download file, you can use the `-be` option to specify a string that can be validated at the remote database, and the `-bg` option to create a download file for new remote databases.

See also

- “`-bc dbmlsync option`” on page 99
- “MobiLink file-based download” [*MobiLink - Server Administration*]

Example

```
CreateDnldFile=dnld1.df
```

DnldFileExtra synchronization profile option

When creating a download file, this option specifies an extra string to be included in the file.

Syntax

be=*string*

DnldFileExtra=*string*

Remarks

The string can be used for authentication or other purposes. It is passed to the `sp_hook_dbmlsync_validate_download_file` stored procedure on the remote database when the download file is applied.

The string may not contain any semicolons.

See also

- “`-be dbmlsync option`” on page 99
- “`sp_hook_dbmlsync_validate_download_file`” on page 229

Example

The following example shows how to use the DnldFileExtra synchronization profile option:

```
DnldFileExtra=val1,val2,val3
```

DownloadOnly synchronization profile option

Perform a download-only synchronization when set to on.

Syntax

ds={ON|OFF}

DownloadOnly={ON|OFF}

Remarks

When download-only synchronization occurs, dbmsync does not upload any database changes. However, it does upload information about the schema and progress offset.

In addition, dbmsync ensures that changes on the remote are not overwritten during download-only synchronization. It does this by scanning the log to detect rows with operations waiting to be uploaded. If any of these rows is modified by the download, the download is rolled back and the synchronization fails. If the synchronization fails for this reason, you must do a full synchronization to correct the problem.

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

See also

- [“-ds dbmsync option” on page 106](#)
- [“DownloadOnly \(ds\) extended option” on page 132](#)

Example

The following example shows how to use the DownloadOnly synchronization profile option:

```
DownloadOnly=on
```

DownloadReadSize synchronization profile option

For restartable downloads, specifies the maximum number of bytes that may need to be resent after a communications failure.

Syntax

drs=size

DownloadReadSize=size

Remarks

Dbmsync reads the download in chunks. The download read size defines the size of these chunks. When a communication error occurs, dbmsync loses the entire chunk that was being processed. Depending on when the error occurs, the number of bytes lost ranges between 0 and the download read size -1. So for example, if the DownloadReadSize is 100 bytes and an error occurs after reading 497 bytes, the last 97 bytes read are lost. Bytes that are lost in this way are resent when the download is restarted.

In general, larger download read size values result in better performance on successful synchronizations but result in more data being resent when an error occurs. The typical use of this option is to reduce the default size when communication is unreliable.

The default is 32767. If you set this option to a value larger than 32767, the value 32767 is used.

You can also specify the download read size using the DownloadReadSize extended option.

See also

- [“-drs dbmlsync option” on page 105](#)
- [“DownloadReadSize \(drs\) extended option” on page 133](#)

Example

The following example shows how to use the DownloadReadSize synchronization profile option:

```
DownloadReadSize=100
```

ExtOpt synchronization profile option

Specifies extended options.

Syntax

```
e={option=value; ...}
```

```
ExtOpt={option=value; ...}
```

Remarks

Extended options can be specified by their long form or short form. See [“MobiLink SQL Anywhere client extended options” on page 126](#).

Extended options are specified in option=value pairs. The list of extended options being set must be enclosed in curly braces {}.

See also

- [“-e dbmlsync option” on page 106](#)

Example

The following example shows how to use the ExtOpt synchronization profile option:

```
ExtOpt={lt=exclusive;tableorder=t1,t2,t3}
```

IgnoreHookErrors synchronization profile option

Ignore errors that occur in hook functions when set to on.

Syntax

```
eh={ON|OFF}
```

```
IgnoreHookErrors={ON|OFF}
```

See also

- [“-eh dbmlsync option” on page 107](#)

Example

The following example shows how to use the IgnoreHookErrors synchronization profile option:

```
IgnoreHookErrors=on
```

IgnoreScheduling synchronization profile option

Ignores the Schedule extended option so that synchronization is immediate.

Syntax

```
is={ON|OFF}
```

```
IgnoreScheduling={ON|OFF}
```

Remarks

This option has a short form and long form: you can use is or IgnoreScheduling.

For information about scheduling, see [“Scheduling synchronization” on page 87](#).

See also

- [“-is dbmlsync option” on page 108](#)

Example

The following example shows how to use the IgnoreScheduling synchronization profile option:

```
IgnoreScheduling=on
```

KillConnections synchronization profile option

Drops conflicting locks to the remote database.

Syntax

```
d={ON|OFF}
```

```
KillConnections={ON|OFF}
```

Remarks

In cases where dbmlsync must obtain locks on the tables being synchronized, if another connection has a lock on one of these tables, the synchronization may fail or be delayed. Specifying this option forces SQL Anywhere to drop any other connections to the remote database that hold conflicting locks so that synchronization can proceed immediately.

See also

- [“-d dbmsync option” on page 103](#)
- [“Concurrency during synchronization” on page 82](#)

Example

The following example shows how to use the KillConnections synchronization profile option:

```
KillConnections=on
```

LogRenameSize synchronization profile option

Renames and restarts the transaction log.

Syntax

```
x=size[ K | M ]
```

```
LogRenameSize=size[ K | M ]
```

Remarks

When this option is set, the transaction log is renamed and restarted during synchronization if it is larger than the specified size in bytes. Use the suffix K or M to specify units of kilobytes or megabytes, respectively.

Set the size to 0 to rename the transaction log regardless of its size.

See also

- [“-x dbmsync option” on page 125](#)

Example

The following example shows how to use the LogRenameSize synchronization profile option:

```
LogRenameSize=512k
```

MobiLinkPwd synchronization profile option

Supplies the password of the MobiLink user.

Syntax

```
mp=password
```

```
MobiLinkPwd=password
```

See also

- [“-mp dbmsync option” on page 110](#)
- [“MobiLinkPwd \(mp\) extended option” on page 140](#)
- [“NewMobiLinkPwd \(mn\) extended option” on page 140](#)
- [“-mn dbmsync option” on page 109](#)

Example

The following example shows how to use the MobiLinkPwd synchronization profile option:

```
MobiLinkPwd=mypassword
```

MLUser synchronization profile option (deprecated)

Specifies the MobiLink user name.

Syntax

```
u=username
```

```
MLUser=username
```

Remarks

This option is deprecated. Use the Subscription synchronization profile option instead.

See also

- [“Subscription synchronization profile option” on page 174](#)
- [“MobiLink users” on page 4](#)
- [“-u dbmlsync option \(deprecated\)” on page 120](#)

Example

The following example shows how to use the MobiLinkPwd synchronization profile option:

```
MLUser=my_user_name
```

NewMobiLinkPwd synchronization profile option

Supplies a new password for the MobiLink user. Use this option when you want to change the existing password.

Syntax

```
mn=new_password
```

```
NewMobiLinkPwd=new_password
```

See also

- [“-mp dbmlsync option” on page 110](#)
- [“-mn dbmlsync option” on page 109](#)
- [“Introduction to MobiLink users” on page 4](#)

Example

The following example shows how to use the NewMobiLinkPwd synchronization profile option:

```
NewMobiLinkPwd=new_password
```

Ping synchronization profile option

Pings a MobiLink server.

Syntax

pi={ON|OFF}

Ping={ON|OFF}

Remarks

When you set the ping synchronization profile option to on, dbmlsync connects to the remote database, retrieves information required to connect to the MobiLink server, connects to the server, and authenticates the specified MobiLink user.

When the MobiLink server receives a ping, it connects to the consolidated database, authenticates the user, and then sends the authenticating user status and value back to the client. If the MobiLink user name cannot be found in the ml_user system table and the MobiLink server is running with the command line option -zu+, the MobiLink server adds the user to the ml_user MobiLink system table.

To adequately test your connection, you should use the ping synchronization profile option with all the synchronization options you want to use to synchronize with dbmlsync. When the ping synchronization profile option is included, dbmlsync does not perform a synchronization.

If the ping succeeds, the MobiLink server issues an information message. If the ping does not succeed, it issues an error message.

See also

- [“-pi dbmlsync option” on page 114](#)

Example

The following example shows how to use the Ping synchronization profile option:

```
Ping=on
```

Publication synchronization profile option

Specifies the publications(s) to synchronize.

Syntax

n=pubname, ...

Publication=pubname, ...

Remarks

The publication synchronization profile option can be specified only once in a synchronization profile.

This option is deprecated. Use the Subscription synchronization profile option instead.

Note

This option has been deprecated. It is recommended that you use the `-s dbmsync` option instead. See [“-s dbmsync option” on page 117](#).

To use the `dbmsync -s` option you need to determine the subscription name for the subscription you want to synchronize. You can determine the subscription name using the following query:

```
SELECT subscription_name
FROM syssync JOIN sys.syspublication
WHERE site_name = <ml_user> AND publication_name = <pub_name>;
```

Replace `<ml_user>` with the MobiLink user you are synchronizing. This is the value specified by the `-u` option on the `dbmsync` command line. See [“-u dbmsync option \(deprecated\)” on page 120](#).

Replace `<pub_name>` with the name of the publication being synchronized. This is the value specified with the `-n` option on the `dbmsync` command line. See [“-n dbmsync option \(deprecated\)” on page 110](#).

See also

- [“-n dbmsync option \(deprecated\)” on page 110](#)

Example

```
Publication=overnight
```

RemoteProgressGreater synchronization profile option

Specifies that the remote offset should be used if it is greater than the consolidated offset. This is equivalent to the `-ra` option.

Syntax

```
ra={ON|OFF}
```

```
RemoteProgressGreater={ON|OFF}
```

Remarks

This option should be used only in very rare cases. If you use this option, the upload is retried starting from the offset obtained from the remote database if the remote offset is greater than the offset obtained from the consolidated database. If you use this option and the offset in the remote is not greater than the offset from the consolidated database, an error is reported and the synchronization is aborted.

This option should be used with care. If the offset mismatch is the result of a restore of the consolidated database, changes that happened in the remote database in the gap between the two offsets are lost. This option may be useful when the consolidated database has been restored from backup and the remote database transaction log has been truncated at the same point as the remote offset. In this case, all data that was uploaded from the remote database is lost from the point of the consolidated offset to the point of the remote offset.

See also

- [“-r dbmlsync option” on page 116](#)

Example

```
RemoteProgressGreater=on
```

RemoteProgressLess synchronization profile option

Specifies that the remote offset should be used if it is less than the consolidated offset (such as when the remote database has been restored from backup). This is equivalent to the -rb option.

Syntax

```
rb={ON|OFF}
```

```
RemoteProgressLess={ON|OFF}
```

Remarks

If the remote database is restored from backup, the default behavior may cause data to be lost. When you use this option, the upload continues from the offset recorded in the remote database if the offset recorded in the remote is less than that obtained from the consolidated database. If you use this option and the offset in the remote is not less than the offset from the consolidated database, an error is reported and the synchronization is aborted.

This option may result in some data being uploaded that has already been uploaded. This can result in conflicts in the consolidated database and should be handled with appropriate conflict resolution scripts.

See also

- [“-r dbmlsync option” on page 116](#)

Example

```
RemoteProgressLess=on
```

Subscription synchronization profile option

Specifies the subscription(s) to synchronize.

Syntax

```
s=pubname, ...
```

```
Subscription=subname, ...
```

Remarks

Subscription can only be specified once in a synchronization profile but the command line option can be specified multiple times.

See also

- [“-s dbmsync option” on page 117](#)

Example

```
Subscription=mySubscription
```

TransactionalUpload synchronization profile option

Boolean. Specifies that each transaction on the remote database should be uploaded as a separate transaction within one synchronization.

Syntax

```
tu={ON|OFF}
```

```
TransactionalUpload={ON|OFF}
```

Remarks

When you use the TransactionalUpload synchronization profile option, you create a transactional upload: dbmsync uploads each transaction on the remote database as a distinct transaction. The MobiLink server applies and commits each transaction separately when it is received.

See also

- [“-tu dbmsync option” on page 119](#)

Example

```
TransactionalUpload=on
```

UpdateGenNum synchronization profile option

When creating a download file, this option creates a file that can be used with remote databases that have not yet synchronize. Otherwise, you must perform a synchronization before you apply a download file.

Syntax

```
bg={ON|OFF}
```

```
UpdateGenNum={ON|OFF}
```

Remarks

This option causes the download file to update the generation numbers on the remote database.

Download files built with this option should be snapshot downloads. Timestamp-based downloads do not work with remote databases that have not synchronized because the last download timestamp on a new remote is by default January 1, 1900, which is earlier than the last download timestamp in the download file. For timestamp-based file-based downloads to work, the last download timestamp in the download file must be the same or earlier than on the remote.

Do not apply UpdateGenNum download files to remote databases that have already synchronized if your system depends on functionality provided by generation numbers as this option circumvents that functionality.

See also

- “-bg dbmlsync option” on page 99
- “MobiLink generation numbers” [*MobiLink - Server Administration*]
- “Synchronizing new remotes” [*MobiLink - Server Administration*]

Example

```
UpdateGenNum=on
```

UploadOnly synchronization profile option

Specifies that synchronization only includes an upload, and that no download should occur.

Syntax

```
uo={ON|OFF}
```

```
UploadOnly={ON|OFF}
```

Remarks

During upload-only synchronization, dbmlsync prepares and sends an upload to MobiLink exactly as it would in a normal full synchronization. However, instead of sending a download back down, MobiLink sends only an acknowledgement indicating if the upload was successfully committed.

For a list of the scripts that must be defined for upload-only synchronization, see “[Required scripts](#)” [*MobiLink - Server Administration*].

See also

- “-uo dbmlsync option” on page 122
- “UploadOnly (uo) extended option” on page 150

Example

```
UploadOnly=on
```

UploadRowCnt synchronization profile option

Integer. Specifies an estimate of the number of rows to be uploaded in a synchronization.

Syntax

```
urc=rowcount
```

```
UploadRowCnt=rowcount
```

Remarks

To improve performance, you can specify an estimate of the number of rows to upload in a synchronization. This setting is especially useful when you are uploading a large number of rows. A higher estimate results in faster uploads but more memory usage.

Synchronization proceeds correctly regardless of the specified estimate.

See also

- [“-urc dbmlsync option” on page 123](#)

Example

```
UploadRowCnt=100
```

Verbosity synchronization profile option

Controls dbmlsync verbosity.

Syntax

```
v={BASIC|HIGH|CONNECT_STR|ROW_CNT|OPTIONS|ML_PASSWORD|ROW_DATA|HOOK}, ...
```

```
Verbosity={BASIC|HIGH|CONNECT_STR|ROW_CNT|OPTIONS|ML_PASSWORD|ROW_DATA|HOOK}, ...
```

Allowed values

The value must be a comma separated list of one or more of the following options, each of which enables a different type of verbosity:

- **BASIC** Generate limited verbosity.
- **HIGH** Generate the maximum possible level of verbosity.
- **CONNECT_STR** Expose the connection string in the log.
- **ROW_CNT** Log the number of rows that were uploaded and downloaded.
- **OPTIONS** Log the options used to specify synchronization.
- **ML_PASSWORD** Expose the MobiLink password in the log.
- **ROW_DATA** Log rows that were uploaded and downloaded.
- **HOOK** Log messages related to hook scripts.

Remarks

If you specify both `-v` and a verbosity extended option and there are conflicts, the `-v` option overrides the extended option. If there is no conflict, the verbosity logging options are additive—all options that you specify are used. When logging verbosity is set by extended option, the logging does not take effect immediately, so startup information is not logged. By the time of the first synchronization, the logging behavior is identical between the `-v` options and the extended options.

See also

- [“-v dbmlsync option” on page 123](#)

Example

```
Verbosity=OPTIONS, ML_PASSWORD
```

Event hooks for SQL Anywhere clients

Introduction to dbmlsync hooks

The SQL Anywhere synchronization client, dbmlsync, provides an optional set of event hooks that you can use to customize the synchronization process. When a hook is implemented, it is called at a specific point in the synchronization process.

You implement an event hook by creating a SQL stored procedure with a specific name. Most event-hook stored procedures are executed on the same connection as the synchronization itself.

You can use event hooks to log and handle synchronization events. For example, you can schedule synchronizations based on logical events, retry connection failures, or handle errors and referential integrity violations.

In addition, you can use event hooks to synchronize subsets of data that cannot be easily defined in a publication. For example, you can synchronize data in a temporary table by writing one event hook procedure to copy data from the temporary table to a permanent table before the synchronization and another to copy the data back afterward.

Caution

The integrity of the synchronization process relies on a sequence of built-in transactions. You must not perform an implicit or explicit commit or rollback within your event-hook procedures.

If you change any connection setting in a hook you must restore the setting to its previous value before the hook ends. Failing to restore the setting may produce unexpected results.

dbmlsync interfaces

You can use client event hooks with the dbmlsync command line utility or any programming interface used to synchronize SQL Anywhere clients, including the dbmlsync API and the DBTools interface for dbmlsync.

See [“Customizing dbmlsync synchronization” on page 89](#).

Synchronization event hook sequence

The following pseudo-code shows the available events and the point at which each is called during the synchronization process. For example, `sp_hook_dbmlsync_abort` is the first event hook to be invoked.

```

sp_hook_dbmlsync_abort //not called when Dbmlsync API or the SQL SYNCHRONIZE
STATEMENT is used
sp_hook_dbmlsync_set_extended_options
loop until return codes direct otherwise (
    sp_hook_dbmlsync_abort
    sp_hook_dbmlsync_delay
)
sp_hook_dbmlsync_abort
// start synchronization
sp_hook_dbmlsync_begin
// upload events
for each upload segment
// a normal synchronization has one upload segment
// a transactional upload has one segment per transaction
// an incremental upload has one segment per upload piece
    sp_hook_dbmlsync_logscan_begin //not called for scripted upload
    sp_hook_dbmlsync_logscan_end //not called for scripted upload
    sp_hook_dbmlsync_set_ml_connect_info //only called during first upload
    sp_hook_dbmlsync_upload_begin
    sp_hook_dbmlsync_set_upload_end_progress //only called for scripted upload
    sp_hook_dbmlsync_upload_end
next upload segment
// download events
sp_hook_dbmlsync_validate_download_file (only called
    when -ba option is used)
sp_hook_dbmlsync_download_begin
for each table
    sp_hook_dbmlsync_download_table_begin
    sp_hook_dbmlsync_download_table_end
next table
sp_hook_dbmlsync_download_end

sp_hook_dbmlsync_schema_upgrade
// end synchronization
sp_hook_dbmlsync_end
sp_hook_dbmlsync_process_exit_code
sp_hook_dbmlsync_log_rescan

```

Event hooks

Each hook is provided with parameter values that you can use when you implement the procedure. Sometimes, you can modify the value to return a new value; others are read-only. These parameters are not stored procedure arguments. No arguments are passed to any of the event-hook stored procedures. Instead, arguments are exchanged by reading and modifying rows in the `#hook_dict` table.

For example, the `sp_hook_dbmlsync_begin` procedure has a MobiLink user parameter, which is the MobiLink user being synchronized. You can retrieve this value from the `#hook_dict` table.

Although the sequence has similarities to the event sequence at the MobiLink server, there is little overlap in the kind of logic you would want to add to the consolidated and remote databases. The two interfaces are therefore separate and distinct.

If a `*_begin` hook executes successfully, the corresponding `*_end` hook is called regardless of any error that occurred after the `*_begin` hook. If the `*_begin` hook is not defined, but you have defined an `*_end` hook, then the `*_end` hook is called unless an error occurs before the point in time where the `*_begin` hook would normally be called.

If the hooks change data in your database, all changes up to and including `sp_hook_dbmsync_logscan_begin` are synchronized in the current synchronization session; after that point, changes are synchronized in the next session.

Using event-hook procedures

This section describes some considerations for designing and using event-hook procedures.

Notes

- Do not perform any COMMIT or ROLLBACK operations in event-hook procedures. The procedures are executed on the same connection as the synchronization, and a COMMIT or ROLLBACK may interfere with synchronization.
- If you change any connection setting in a hook you must restore the setting to its previous value before the hook ends. Failing to restore the setting may produce unexpected results.
- Dbmsync calls the stored procedures without qualifying them by owner. The stored procedures must therefore be owned by either the user name employed on the dbmsync connection, or a group of which that user is a member.
- Hook procedures must be created by a user with DBA authority.

#hook_dict table

Immediately before a hook is called, dbmsync creates the #hook_dict table in the remote database, using the following CREATE statement. The # before the table name means that the table is temporary.

```
CREATE TABLE #hook_dict(  
  name VARCHAR(128) NOT NULL UNIQUE,  
  value VARCHAR(10240) NOT NULL)
```

The dbmsync utility uses the #hook_dict table to pass values to hook functions, and hook functions use the #hook_dict table to pass values back to dbmsync.

Each hook receives parameter values. Sometimes, you can modify the value to return a new value; others are read-only. Each row in the table contains the value for one parameter.

For example, suppose two subscriptions are defined as follows:

```
CREATE SYNCHRONIZATION SUBSCRIPTION sub1  
  TO pub1  
  FOR MyUser;  
  SCRIPT VERSION 'v1'  
  
CREATE SYNCHRONIZATION SUBSCRIPTION sub2  
  TO pub2  
  FOR MyUser;  
  SCRIPT VERSION 'v1'
```

When the `sp_hook_dbmsync_begin` hook is called for the following dbmsync command line

```
dbmlsync -c 'dsn=MyDsn' -s sub1,sub2
```

the #hook_dict table contains the following rows:

#hook_dict row	Value
subscription_0	sub1
subscription_1	sub2
publication_0	pub1
publication_1	pub2
MobiLink user	MyUser
Script version	v1

Note

The publication_n rows are deprecated and may be removed in a future release.

A hook can retrieve values from the #hook_dict table and use them to customize behavior. For example, to retrieve the MobiLink user you would use a SELECT statement like this:

```
SELECT value
FROM #hook_dict
WHERE name = 'MobiLink user'
```

In/out parameters can be updated by your hook to modify the behavior of dbmlsync. For example, in the sp_hook_dbmlsync_abort hook you could instruct dbmlsync to abort synchronization by updating the abort synchronization row of the table using a statement like this:

```
UPDATE #hook_dict
SET value='true'
WHERE name='abort synchronization'
```

The description of each hook lists the rows in the #hook_dict table.

Examples

The following sample sp_hook_dbmlsync_delay procedure illustrates the use of in/out parameters in the #hook_dict table. The procedure allows synchronization only outside a scheduled down time of the MobiLink system between 18:00 and 19:00.

```
CREATE PROCEDURE sp_hook_dbmlsync_delay()
BEGIN
  DECLARE delay_val integer;
  SET delay_val=DATEDIFF(
    second, CURRENT TIME, '19:00');
  IF (delay_val>0 AND
    delay_val<3600)
  THEN
    UPDATE #hook_dict SET value=delay_val
    WHERE name='delay duration';
```

```
END IF;  
END
```

The following procedure is executed in the remote database at the beginning of synchronization. It retrieves the current MobiLink user name (one of the parameters available for the `sp_hook_dbmlsync_begin` event), and displays it on the SQL Anywhere messages window.

```
CREATE PROCEDURE sp_hook_dbmlsync_begin()  
BEGIN  
    DECLARE MLuser VARCHAR(150);  
    SELECT '>>>MLuser = ' || value INTO MLuser  
        FROM #hook_dict  
        WHERE name = 'MobiLink user name';  
    MESSAGE syncdef TYPE INFO TO CONSOLE;  
END
```

Connections for event-hook procedures

Each event-hook procedure is executed on the same connection as the synchronization itself. The following are exceptions:

- `sp_hook_dbmlsync_all_error`
- `sp_hook_dbmlsync_communication_error`
- `sp_hook_dbmlsync_download_log_ri_violation`
- `sp_hook_dbmlsync_misc_error`
- `sp_hook_dbmlsync_sql_error`

These procedures are called before a synchronization fails. On failure, synchronization actions are rolled back. By executing on a separate connection, you can use these procedures to log information about the failure, without the logging actions being rolled back along with the synchronization actions.

Handling errors and warnings in event hook procedures

You can create event hook stored procedures to handle synchronization errors, MobiLink connection failures, and referential integrity violations. This section describes event hook procedures that are used to handle errors and warnings. Once implemented, each procedure is automatically executed whenever an error of the named type occurs.

Handling RI violations

Referential integrity violations occur when rows in the download violate foreign key relationships on the remote database. Use the following event hooks to log and handle referential integrity violations:

- [“sp_hook_dbmlsync_download_log_ri_violation” on page 196](#)
- [“sp_hook_dbmlsync_download_ri_violation” on page 198](#)

Handling MobiLink connection failures

The `sp_hook_dbmsync_ml_connect_failed` event hook allows you to retry failed attempts to connect to the MobiLink server using a different communication type or address. If connection ultimately fails, `dbmsync` calls the `sp_hook_dbmsync_communication_error` and `sp_hook_dbmsync_all_error` hooks.

See [“sp_hook_dbmsync_ml_connect_failed” on page 213](#).

Handling dbmsync errors

Each time a `dbmsync` error message is generated, the following hooks are called:

- First, depending on the type of error, one of the following hooks is called: `sp_hook_dbmsync_communication_error`, `sp_hook_dbmsync_misc_error`, or `sp_hook_dbmsync_sql_error`. These hooks contain information specific to the type of error; for example, `sqlcode` and `sqlstate` are provided for SQL errors.
- Next, `sp_hook_dbmsync_all_error` is called. This hook is useful for logging all errors that occur.

See:

- [“sp_hook_dbmsync_communication_error” on page 189](#)
- [“sp_hook_dbmsync_sql_error” on page 224](#)
- [“sp_hook_dbmsync_misc_error” on page 210](#)
- [“sp_hook_dbmsync_all_error” on page 185](#)

If you want to restart a synchronization in response to an error, you can use the user state parameter in `sp_hook_dbmsync_end`.

See [“sp_hook_dbmsync_end” on page 203](#).

Ignoring errors

By default, synchronization stops when an unhandled error is encountered in an event hook procedure. You can instruct the `dbmsync` utility to ignore these errors by supplying the `-eh` option.

See [“IgnoreHookErrors \(eh\) extended option” on page 136](#).

sp_hook_dbmsync_abort

Use this stored procedure to cancel the synchronization process.

Rows in #hook_dict table

Name	Value	Description
abort synchronization (in out)	true false	If you set the abort synchronization row of the <code>#hook_dict</code> table to true , then synchronization terminates immediately after the event.

Name	Value	Description
publication_ <i>n</i> (in)	publication	Deprecated. Use subscription_ <i>n</i> instead. The publications being synchronized, where <i>n</i> is an integer. There is one publication_ <i>n</i> entry for each publication being synchronized. The numbering of <i>n</i> starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
exit code (in out)	number	When abort synchronization is set to TRUE, you can use this value to set the exit code for the aborted synchronization. 0 indicates a successful synchronization. Any other number indicates that the synchronization failed.
script version (in out)	script version name	The MobiLink script version to be used for the synchronization.
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. There is one subscription_ <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

If a procedure of this name exists, it is called at dbmlsync startup, and then again after each synchronization delay that is caused by the sp_hook_dbmlsync_delay hook.

When dbmlsync is run from the command line, setting the abort synchronization to true causes all remaining synchronizations (including scheduled synchronizations) to be canceled. When the dbmlsync API or the SQL SYNCHRONIZE statement is used, setting abort synchronization to true only causes the current synchronization to be aborted.

Actions of this procedure are committed immediately after execution.

See also

- [“Synchronization event hook sequence” on page 178](#)
- [“sp_hook_dbmlsync_process_exit_code” on page 216](#)

Examples

The following procedure prevents synchronization during a scheduled maintenance hour between 19:00 and 20:00 each day.

```
CREATE PROCEDURE sp_hook_dbmlsync_abort()
BEGIN
  DECLARE down_time_start TIME;
  DECLARE is_down_time VARCHAR(128);
```

```

SET down_time_start='19:00';
IF datediff( hour,down_time_start,now(*) ) < 1
THEN
  set is_down_time='true';
ELSE
  SET is_down_time='false';
END IF;
UPDATE #hook_dict
SET value = is_down_time
WHERE name = 'abort synchronization'
END;

```

Suppose you have an abort hook that may abort synchronization for one of two reasons. One of the reasons indicates normal completion of synchronization, so you want dbmsync to have an exit code of 0. The other reason indicates an error condition, so you want dbmsync to have a non-zero exit code. You could achieve this with an `sp_hook_dbmsync_abort` hook defined as follows.

```

BEGIN
  IF [condition that defines the normal abort case] THEN
    UPDATE #hook_dict SET value = '0'
    WHERE name = 'exit code';
    UPDATE #hook_dict SET value = 'TRUE'
    WHERE name = 'abort synchronization';
  ELSEIF [condition that defines the error abort case] THEN
    UPDATE #hook_dict SET value = '1'
    WHERE name = 'exit code';
    UPDATE #hook_dict SET value = 'TRUE'
    WHERE name = 'abort synchronization';
  END IF;
END;

```

sp_hook_dbmsync_all_error

Use this stored procedure to process dbmsync error messages of all types. For example, you can implement the `sp_hook_dbmsync_all_error` hook to log errors or perform a specific action when a specific error occurs.

Rows in #hook_dict table

Name	Value	Description
publication_n (in)	publication	Deprecated. Use subscription_n instead. The publications being synchronized, where n is an integer. There is one publication_n entry for each publication being synchronized. The numbering of n starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version that is used for the synchronization.

Name	Value	Description
error message (in)	error message text	This is the same text that is displayed in the dbmlsync log.
error id (in)	integer	An ID that uniquely identifies the message. Use this row to identify the error message, as the error message text may change.
error hook user state (in/out)	integer	This value can be set by the hook to pass state information to future calls to the <code>sp_hook_dbmlsync_all_error</code> , <code>sp_hook_dbmlsync_communication_error</code> , <code>sp_hook_dbmlsync_misc_error</code> , <code>sp_hook_dbmlsync_sql_error</code> , or <code>sp_hook_dbmlsync_end</code> hooks. The first time one of these hooks is called, the value of the row is 0. If a hook changes the value of the row, the new value is used in the next hook call.
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_ <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

Each time a dbmlsync error message is generated, the following hooks are called:

- First, depending on the type of error, one of the following hooks is called: `sp_hook_dbmlsync_communication_error`, `sp_hook_dbmlsync_misc_error`, or `sp_hook_dbmlsync_sql_error`. These hooks contain information specific to the type of error; for example, `sqlcode` and `sqlstate` are provided for SQL errors.
- Next, the `sp_hook_dbmlsync_all_error` is called. This hook is useful for logging all errors that occurred.

If an error occurs during startup before a synchronization has been initiated, the `#hook_dict` entries for MobiLink user and Script version are set to an empty string, and no `publication_n` or `subscription_n` rows are set in the `#hook_dict` table.

The error hook user state row provides a useful mechanism for you to pass information about the nature of the error to the `sp_hook_dbmlsync_end` hook, where you might use that information to decide whether to retry the synchronization.

This procedure executes on a separate connection to ensure that operations it performs are not lost if a rollback is performed on the synchronization connection. If dbmlsync cannot establish a separate connection, the procedure is not called.

Since this hook executes on a separate connection you should use care when accessing tables that are being synchronized in your hook procedure because dbmlsync may have locks on these tables. These locks could cause operations in your hook to fail or to wait indefinitely.

Actions of this procedure are committed immediately after the hook completes.

See also

- [“Handling errors and warnings in event hook procedures” on page 182](#)
- [“sp_hook_dbmlsync_communication_error” on page 189](#)
- [“sp_hook_dbmlsync_misc_error” on page 210](#)
- [“sp_hook_dbmlsync_sql_error” on page 224](#)

Example

Assume you use the following table to log errors in the remote database.

```
CREATE TABLE error_log
(
  pk INTEGER DEFAULT AUTOINCREMENT PRIMARY KEY,
  err_id INTEGER,
  err_msg VARCHAR(10240),
);
```

The following example sets up sp_hook_dbmlsync_all_error to log errors.

```
CREATE PROCEDURE sp_hook_dbmlsync_all_error()
BEGIN
  DECLARE msg VARCHAR(10240);
  DECLARE id INTEGER;

  // get the error message text
  SELECT value INTO msg
  FROM #hook_dict
  WHERE name = 'error message';

  // get the error id
  SELECT value INTO id
  FROM #hook_dict
  WHERE name = 'error id';

  // log the error information
  INSERT INTO error_log(err_msg, err_id)
  VALUES (msg, id);
END;
```

To see possible error id values, test run dbmlsync. For example, if dbmlsync returns the error "Unable to connect to MobiLink server", sp_hook_dbmlsync_all_error inserts the following row in error_log.

```
1,14173,
'Unable to connect to MobiLink server'
```

Now, you can associate the error "Unable to connect to MobiLink server" with the error id 14173.

The following example sets up hooks to retry the synchronization whenever error 14173 occurs.

```
CREATE PROCEDURE sp_hook_dbmlsync_all_error()
BEGIN
  IF EXISTS( SELECT value FROM #hook_dict
```

```

        WHERE name = 'error id' AND value = '14173' )
    THEN
        UPDATE #hook_dict SET value = '1'
            WHERE name = 'error hook user state';
    END IF;
END;

CREATE PROCEDURE sp_hook_dbmlsync_end()
BEGIN
    IF EXISTS( SELECT value FROM #hook_dict
        WHERE name='error hook user state' AND value='1')
    THEN
        UPDATE #hook_dict SET value = 'sync'
            WHERE name='restart';
    END IF;
END;

```

See “[sp_hook_dbmlsync_end](#)” on page 203.

sp_hook_dbmlsync_begin

Use this stored procedure to add custom actions at the beginning of the synchronization process.

Rows in #hook_dict table

Name	Value	Description
publication_ <i>n</i> (in)	publication	Deprecated. Use subscription_ <i>n</i> instead. The publications being synchronized, where <i>n</i> is an integer. There is one publication_ <i>n</i> entry for each publication being synchronized. The numbering of <i>n</i> starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_ <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

If a procedure of this name exists, it is called at the beginning of the synchronization process.

Actions of this procedure are committed immediately after execution.

See also

- [“Synchronization event hook sequence” on page 178](#)

Examples

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
  "event_id"          integer NOT NULL DEFAULT autoincrement ,
  "event_time"       timestamp NULL,
  "event_name"       varchar(128) NOT NULL ,
  "subs"             varchar(1024) NULL ,
  PRIMARY KEY ("event_id")
)
```

The following logs the beginning of each synchronization in the table.

```
CREATE PROCEDURE sp_hook_dbmlsync_begin ()
BEGIN

  DECLARE subs_list VARCHAR(1024);

  -- build a list of subscriptions being synchronized
  SELECT LIST(value) INTO subs_list
  FROM #hook_dict
  WHERE name LIKE 'subscription_%';

  -- log the event
  INSERT INTO SyncLog(event_time, event_name, subs)
  VALUES( CURRENT_TIMESTAMP, 'sp_hook_dbmlsync_begin', subs_list );
END
```

sp_hook_dbmlsync_communication_error

Use this stored procedure to process communications errors.

Rows in #hook_dict table

Name	Value	Description
publication_ <i>n</i> (in)	publication	Deprecated. Use subscription_ <i>n</i> instead. The publications being synchronized, where <i>n</i> is an integer. There is one publication_ <i>n</i> entry for each publication being synchronized. The numbering of <i>n</i> starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version that is used for the synchronization.

Name	Value	Description
error message (in)	error message text	This is the same text that is displayed in the dbmlsync log.
error id (in)	numeric	An ID that uniquely identifies the message. Use this row to identify the error message, as the error message text may change.
error hook user state (in out)	integer	This value can be set by the hook to pass state information to future calls to the sp_hook_dbmlsync_all_error, sp_hook_dbmlsync_communication_error, sp_hook_dbmlsync_misc_error, sp_hook_dbmlsync_sql_error, or sp_hook_dbmlsync_end hooks. The first time one of these hooks is called, the value of the row is 0. If a hook changes the value of the row, the new value is used in the next hook call.
stream error code (in)	integer	The error reported by the stream.
system error code (in)	integer	A system-specific error code.
subscription_n (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_n entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

If an error occurs during startup before a synchronization has been initiated, the #hook_dict entries for MobiLink user and Script version are set to an empty string, and no publication_n or subscription_nrows are set in the #hook_dict table.

When communication errors occur between dbmlsync and the MobiLink server, this hook allows you to access stream-specific error information.

The **stream error code** parameter is an integer indicating the type of communication error.

For a listing of possible error code values, see [“MobiLink communication error messages” \[Error Messages\]](#).

The error hook user state row provides a useful mechanism for you to pass information about the nature of the error to the sp_hook_dbmlsync_end hook, where you might use that information to decide whether to retry the synchronization.

This procedure executes on a separate connection to ensure that operations it performs are not lost if a rollback is performed on the synchronization connection. If dbmsync cannot establish a separate connection, the procedure is not called.

Since this hook executes on a separate connection you should use care when accessing tables that are being synchronized in your hook procedure because dbmsync may have locks on these tables. These locks could cause operations in your hook to fail or to wait indefinitely.

Actions of this procedure are committed immediately after execution.

See also

- [“Handling errors and warnings in event hook procedures” on page 182](#)
- [“sp_hook_dbmsync_all_error” on page 185](#)
- [“sp_hook_dbmsync_misc_error” on page 210](#)
- [“sp_hook_dbmsync_sql_error” on page 224](#)

Example

Assume you use the following table to log communication errors in the remote database.

```
CREATE TABLE communication_error_log
(
  error_msg VARCHAR(10240),
  error_code VARCHAR(128)
);
```

The following example sets up sp_hook_dbmsync_communication_error to log communication errors.

```
CREATE PROCEDURE sp_hook_dbmsync_communication_error()
BEGIN
  DECLARE msg VARCHAR(255);
  DECLARE code INTEGER;

  // get the error message text
  SELECT value INTO msg
  FROM #hook_dict
  WHERE name = 'error message';

  // get the error code
  SELECT value INTO code
  FROM #hook_dict
  WHERE name = 'stream error code';

  // log the error information
  INSERT INTO communication_error_log(error_code,error_msg)
  VALUES (code,msg);
END
```

sp_hook_dbmsync_delay

Use this stored procedure to control when synchronization takes place.

Rows in #hook_dict table

Name	Value	Description
delay duration (in out)	number of seconds	If the procedure sets the delay duration value to zero, then dbmlsync synchronization proceeds immediately. A non-zero delay duration value specifies the number of seconds before the delay hook is called again.
maximum accumulated delay (in out)	number of seconds	The maximum accumulated delay specifies the maximum number of seconds delay before each synchronization. Dbmlsync keeps track of the total delay created by all calls to the delay hook since the last synchronization. If no synchronization has occurred since dbmlsync started running, the total delay is calculated from the time dbmlsync started up. When the total delay exceeds the value of maximum accumulated delay, synchronization begins without any further calls to the delay hook.
publication_ <i>n</i> (in)	publication	Deprecated. Use subscription_ <i>n</i> instead. The publications being synchronized, where <i>n</i> is an integer. There is one publication_ <i>n</i> entry for each publication being synchronized. The numbering of <i>n</i> starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_ <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

If a procedure of this name exists, it is called before **sp_hook_dbmlsync_begin** at the beginning of the synchronization process.

This hook is not called when synchronization is initiated using the Dbmlsync API or the SQL SYNCHRONIZE statement.

Actions of this procedure are committed immediately after execution.

See also

- [“Synchronization event hook sequence” on page 178](#)
- [“Initiating synchronization with event hooks” on page 88](#)
- [“sp_hook_dbmsync_download_end” on page 195](#)

Example

Assume you have the following table to log orders on the remote database.

```
CREATE TABLE OrdersTable(
  "id" INTEGER PRIMARY KEY DEFAULT AUTOINCREMENT,
  "priority" VARCHAR(128)
);
```

The following example delays synchronization for a maximum accumulated delay of one hour. Every ten seconds the hook is called again and checks for a high priority row in the OrdersTable. If a high priority row exists, the delay duration is set to zero to start the synchronization process.

```
CREATE PROCEDURE sp_hook_dbmsync_delay()
BEGIN
  -- Set the maximum delay between synchronizations
  -- or before the first synchronization starts to 1 hour
  UPDATE #hook_dict SET value = '3600' // 3600 seconds
  WHERE name = 'maximum accumulated delay';

  -- check if a high priority order exists in OrdersTable
  IF EXISTS (SELECT * FROM OrdersTable where priority='high') THEN
  -- start the synchronization to process the high priority row
  UPDATE #hook_dict
  SET value = '0'
  WHERE name='delay duration';
  ELSE
  -- set the delay duration to call this procedure again
  -- following a 10 second delay
  UPDATE #hook_dict
  SET value = '10'
  WHERE name='delay duration';
  END IF;
END;
```

In the sp_hook_dbmsync_end hook you can mark the high priority row as processed:

```
CREATE PROCEDURE sp_hook_dbmsync_upload_end()
BEGIN
  IF EXISTS( SELECT value FROM #hook_dict
  WHERE name = 'Upload status'
  AND value = 'committed' ) THEN
  UPDATE OrderTable SET priority = 'high-processed'
  WHERE priority = 'high';
  END IF;
END;
```

This example assumes that you have used LockTables extended option to ensure that the tables are locked during synchronization. If the tables are not locked, it is possible for a high priority row to be inserted after the upload is built but before the sp_hook_dbmsync_end hook is executed. If that happened the row's priority would be changed to "high-processed" even though it was never uploaded.

See [“sp_hook_dbmsync_end” on page 203](#).

sp_hook_dbmlsync_download_begin

Use this stored procedure to add custom actions at the beginning of the download stage of the synchronization process.

Rows in #hook_dict table

Name	Value	Description
publication_ <i>n</i> (in)	publication	Deprecated. Use subscription_ <i>n</i> instead. The publications being synchronized, where <i>n</i> is an integer. There is one publication_ <i>n</i> entry for each publication being synchronized. The numbering of <i>n</i> starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_ <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

If a procedure of this name exists, it is called at the beginning of the download stage of the synchronization process.

Actions of this procedure are committed or rolled back when the download is committed or rolled back.

See also

- [“Synchronization event hook sequence” on page 178](#)

Example

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
  "event_id"          integer NOT NULL DEFAULT autoincrement ,
  "event_time"        timestamp NULL,
  "event_name"        varchar(128) NOT NULL ,
  "subs"              varchar(1024) NULL ,
  PRIMARY KEY ("event_id")
)
```

The following logs the beginning of the download for each synchronization.

```
CREATE PROCEDURE sp_hook_dbmlsync_download_begin ()
BEGIN
```

```

    DECLARE subs_list VARCHAR(1024);

    -- build a list of subscriptions being synchronized
    SELECT LIST(value) INTO subs_list
    FROM #hook_dict
    WHERE name LIKE 'subscription_%';

    -- log the event
    INSERT INTO SyncLog(event_time, event_name, subs)
    VALUES( CURRENT_TIMESTAMP, 'sp_hook_dbmlsync_download_begin',
    subs_list );
END

```

sp_hook_dbmlsync_download_end

Use this stored procedure to add custom actions at the end of the download stage of the synchronization process.

Rows in #hook_dict table

Name	Value	Description
publication_n (in)	publication	Deprecated. Use subscription_n instead. The publications being synchronized, where <i>n</i> is an integer. There is one publication_n entry for each publication being synchronized. The numbering of <i>n</i> starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
subscription_n (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_n entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

If a procedure of this name exists, it is called at the end of the download stage of the synchronization process.

Actions of this procedure are committed or rolled back when the download is committed or rolled back.

See also

- [“Synchronization event hook sequence” on page 178](#)
- [“Initiating synchronization with event hooks” on page 88](#)
- [“sp_hook_dbmsync_delay” on page 191](#)

Examples

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
  "event_id"          integer NOT NULL DEFAULT autoincrement ,
  "event_time"        timestamp NULL,
  "event_name"        varchar(128) NOT NULL ,
  "subs"              varchar(1024) NULL ,
  PRIMARY KEY ("event_id")
)
```

The following logs the end of the download for each synchronization.

```
CREATE PROCEDURE sp_hook_dbmsync_download_end ()
BEGIN
  DECLARE subs_list VARCHAR(1024);

  -- build a list of subscriptions being synchronized
  SELECT LIST(value) INTO subs_list
  FROM #hook_dict
  WHERE name LIKE 'subscription_%';

  -- log the event
  INSERT INTO SyncLog(event_time, event_name, subs)
  VALUES( CURRENT_TIMESTAMP, 'sp_hook_dbmsync_download_end', subs_list );
END
```

sp_hook_dbmsync_download_log_ri_violation

Logs referential integrity violations in the download process.

Rows in #hook_dict table

Name	Value	Description
publication_n (in)	publication	Deprecated. Use subscription_n instead. The publications being synchronized, where n is an integer. There is one publication_n entry for each publication being synchronized. The numbering of n starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.

Name	Value	Description
foreign key table (in)	table name	The table containing the foreign key column for which the hook is being called.
primary key table (in)	table name	The table referenced by the foreign key for which the hook is being called.
role name (in)	role name	The role name of the foreign key for which the hook is being called.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_ <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

A download RI violation occurs when rows in the download violate foreign key relationships on the remote database. This hook allows you to log RI violations as they occur so that you can investigate their cause later.

After the download is complete, but before it is committed, dbmsync checks for RI violations. If it finds any, it identifies a foreign key that has an RI violation and calls `sp_hook_dbmsync_download_log_ri_violation` (if it is implemented). It then calls `sp_hook_dbmsync_download_ri_violation` (if it is implemented). If there is still a conflict, dbmsync deletes the rows that violate the foreign key constraint. This process is repeated for remaining foreign keys that have RI violations.

This hook is called only when there are RI violations involving tables that are currently being synchronized. If there are RI violations involving tables that are not being synchronized, this hook is not called and the synchronization fails.

This hook is called on a separate connection from the one that dbmsync uses for the download. The connection used by the hook has an isolation level of 0 so that the hook can see the rows that have been applied from the download that are not yet committed. The actions of the hook are committed immediately after it completes so that changes made by this hook are preserved regardless of whether the download is committed or rolled back.

Since this hook executes on a separate connection you should use care when accessing tables that are being synchronized in your hook procedure because dbmsync may have locks on these tables. These locks could cause operations in your hook to fail or to wait indefinitely.

Do not attempt to use this hook to correct RI violation problems. It should be used for logging only. Use `sp_hook_dbmsync_download_ri_violation` to resolve RI violations.

See also

- [“sp_hook_dbmsync_download_ri_violation” on page 198](#)
- [“Synchronization event hook sequence” on page 178](#)

Examples

Assume you use the following table to log referential integrity violations.

```
CREATE TABLE DBA.LogRIViolationTable
(
    entry_time    TIMESTAMP,
    pk_table      VARCHAR( 255 ),
    fk_table      VARCHAR( 255 ),
    role_name     VARCHAR( 255 )
);
```

The following example logs the foreign key table name, primary key table name, and role name when a referential integrity violation is detected on the remote database. The information is stored in LogRIViolationTable on the remote database.

```
CREATE PROCEDURE sp_hook_dbmsync_download_log_ri_violation()
BEGIN
    INSERT INTO DBA.LogRIViolationTable VALUES(
        CURRENT_TIMESTAMP,
        (SELECT value FROM #hook_dict WHERE name = 'Primary key table'),
        (SELECT value FROM #hook_dict WHERE name = 'Foreign key table'),
        (SELECT value FROM #hook_dict WHERE name = 'Role name' ) );
END;
```

sp_hook_dbmsync_download_ri_violation

Allows you to resolve referential integrity violations in the download process.

Rows in #hook_dict table

Name	Value	Description
publication_n (in)	publication	Deprecated. Use subscription_n instead. The publications being synchronized, where n is an integer. There is one publication_n entry for each publication being synchronized. The numbering of n starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
foreign key table (in)	table name	The table containing the foreign key column for which the hook is being called.
primary key table (in)	table name	The table referenced by the foreign key for which the hook is being called.

Name	Value	Description
role name (in)	role name	The role name of the foreign key for which the hook is being called.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_ <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

A download RI violation occurs when rows in the download violate foreign key relationships on the remote database. This hook allows you to attempt to resolve RI violations before dbmsync deletes the rows that are causing the conflict.

After the download is complete, but before it is committed, dbmsync checks for RI violations. If it finds any, it identifies a foreign key that has an RI violation and calls `sp_hook_dbmsync_download_log_ri_violation` (if it is implemented). It then calls `sp_hook_dbmsync_download_ri_violation` (if it is implemented). If there is still a conflict, dbmsync deletes the rows. This process is repeated for remaining foreign keys that have RI violations.

This hook is called only when there are RI violations involving tables that are currently being synchronized. If there are RI violations involving tables that are not being synchronized, this hook is not called and the synchronization fails.

This hook is called on the same connection that dbmsync uses for the download. This hook should not contain any explicit or implicit commits, because they may lead to inconsistent data in the database. The actions of this hook are committed or rolled back when the download is committed or rolled back.

Unlike other hook actions, the operations performed during this hook are not uploaded during the next synchronization.

See also

- [“sp_hook_dbmsync_download_log_ri_violation” on page 196](#)

Example

This example uses the Department and Employee tables shown below:

```
CREATE TABLE Department(
  "department_id"  INTEGER primary key
);

CREATE TABLE Employee(
  "employee_id"   INTEGER PRIMARY KEY,
  "department_id" INTEGER,
```

```
FOREIGN KEY EMPLOYEE_FK1 (department_id) REFERENCES Department
);
```

The following `sp_hook_dbmlsync_download_ri_violation` definition cleans up referential integrity violations between the `Department` and `Employee` tables. It verifies the role name for the foreign key and inserts missing `department_id` values into the `Department` table.

```
CREATE PROCEDURE sp_hook_dbmlsync_download_ri_violation()
BEGIN

IF EXISTS (SELECT * FROM #hook_dict WHERE name = 'role name'
AND value = 'EMPLOYEE_FK1') THEN

-- update the Department table with missing department_id values
INSERT INTO Department
SELECT distinct department_id FROM Employee
WHERE department_id NOT IN (SELECT department_id FROM Department)

END IF;
END;
```

sp_hook_dbmlsync_download_table_begin

Use this stored procedure to add custom actions immediately before each table is downloaded.

Rows in #hook_dict table

Name	Value	Description
table name (in)	table name	The table to which operations are about to be applied.
publication_n (in)	publication	Deprecated. Use <code>subscription_n</code> instead. The publications being synchronized, where <i>n</i> is an integer. There is one <code>publication_n</code> entry for each publication being synchronized. The numbering of <i>n</i> starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
subscription_n (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one <code>subscription_n</code> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

If a procedure of this name exists, it is called for each table immediately before downloaded operations are applied to that table. Actions of this procedure are committed or rolled back when the download is committed or rolled back.

See also

- [“Synchronization event hook sequence” on page 178](#)

Examples

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
  "event_id"          integer NOT NULL DEFAULT autoincrement ,
  "event_time"        timestamp NULL,
  "event_name"        varchar(128) NOT NULL ,
  "subs"              varchar(1024) NULL ,
  PRIMARY KEY ("event_id")
)
```

The following logs the beginning of each table's download for each synchronization.

```
CREATE PROCEDURE sp_hook_dbmsync_download_table_begin ()
BEGIN

  DECLARE subs_list VARCHAR(1024);

  -- build a list of subscriptions being synchronized
  SELECT LIST(value) INTO subs_list
  FROM #hook_dict
  WHERE name LIKE 'subscription_%';

  -- log the event
  INSERT INTO SyncLog(event_time, event_name, subs)
  VALUES( CURRENT TIMESTAMP, 'sp_hook_dbmsync_download_table_begin,
subs_list );
END
```

sp_hook_dbmsync_download_table_end

Use this stored procedure to add custom actions immediately after each table is downloaded.

Rows in #hook_dict table

Name	Value	Description
table name (in)	table name	The table to which operations have just been applied.
delete count (in)	number of rows	The number of rows in this table deleted by the download.
upsert count (in)	number of rows	The number of rows in this table updated or inserted by the download.

Name	Value	Description
publication_ <i>n</i> (in)	publication	Deprecated. Use subscription_ <i>n</i> instead. The publications being synchronized, where <i>n</i> is an integer. There is one publication_ <i>n</i> entry for each publication being synchronization. The numbering of <i>n</i> starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_ <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

If a procedure of this name exists, it is called immediately after all operations in the download for a table have been applied.

Actions of this procedure are committed or rolled back when the download is committed or rolled back.

See also

- [“Synchronization event hook sequence” on page 178](#)

Examples

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
  "event_id"          integer NOT NULL DEFAULT autoincrement ,
  "event_time"       timestamp NULL,
  "event_name"       varchar(128) NOT NULL ,
  "subs"             varchar(1024) NULL ,
  PRIMARY KEY ("event_id")
)
```

The following logs the end of the download for each table for each synchronization.

```
CREATE PROCEDURE sp_hook_dbmlsync_download_table_end ()
BEGIN

  DECLARE subs_list VARCHAR(1024);

  -- build a list of subscriptions being synchronized
  SELECT LIST(value) INTO subs_list
  FROM #hook_dict
  WHERE name LIKE 'subscription_%';
```

```

-- log the event
  INSERT INTO SyncLog(event_time, event_name, subs)
  VALUES( CURRENT_TIMESTAMP, 'sp_hook_dbmsync_download_table_end,
subs_list );
END

```

sp_hook_dbmsync_end

Use this stored procedure to add custom actions immediately before synchronization is complete.

Rows in #hook_dict table

Name	Value	Description
restart (out)	sync download none	<p>If set to sync, then dbmsync retries the synchronization it just completed. The value sync replaces true, which is identical but is deprecated.</p> <p>If set to none (the default), then dbmsync shuts down or restarts according to its command line arguments. The value none replaces false, which is identical but is deprecated.</p> <p>If set to download and the restartable download parameter is true, then dbmsync attempts to restart the download that just failed.</p>
exit code (in)	number	The exit code for the synchronization just completed. A value other than zero represents a synchronization error.
publication_n (in)	publication	Deprecated. Use subscription_n instead. The publications being synchronized, where <i>n</i> is an integer. There is one publication_n entry for each publication being synchronized. The numbering of <i>n</i> starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.

Name	Value	Description
upload status (in)	not sent committed failed unknown	<p>Specifies the status returned by the MobiLink server when dbmsync attempted to verify receipt of the upload. The status can be:</p> <ul style="list-style-type: none"> ● not sent - No upload was sent to the MobiLink server, either because an error prevented it or because the requested synchronization did not require it. This can occur during a download-only synchronization, a restarted download, or a file-based download. ● committed - The upload was received by the MobiLink server, and committed. ● failed - The MobiLink server did not commit the upload. For a transactional upload, the upload status is 'failed' when some but not all the transactions were successfully uploaded and acknowledged by the server. ● unknown - The upload was not acknowledged by the MobiLink server. There is no way to know if it was committed or not.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
restartable download (in)	true/false	If true , the download for the current synchronization failed and can be restarted. If false , the download was successful or it cannot be restarted.
restartable download size (in)	integer	When the restartable download parameter is true , this parameter indicates the number of bytes that were received before the download failed. When restartable download is false , this value is meaningless.
error hook user state (in)	integer	This value contains information about errors and can be sent from the hooks <code>sp_hook_dbmsync_all_error</code> , <code>sp_hook_dbmsync_communication_error</code> , <code>sp_hook_dbmsync_misc_error</code> , or <code>sp_hook_dbmsync_sql_error</code> .

Name	Value	Description
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_ <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

If a procedure of this name exists, it is called at the end of each synchronization.

If an `sp_hook_dbmsync_end` hook is defined so that the hook always sets the restart parameter to **sync**, and you specify multiple subscriptions on the `dbmsync` command line in the form `-s sub1, -s sub2`, and so on, then `dbmsync` repeatedly synchronizes the first publication and never synchronizes the second.

Actions of this procedure are committed immediately after execution.

See also

- [“Introduction to dbmsync hooks”](#) on page 178
- [“Synchronization event hook sequence”](#) on page 178
- [“Resuming failed downloads”](#) [*MobiLink - Server Administration*]
- [“Handling errors and warnings in event hook procedures”](#) on page 182

Examples

In the following example the download is manually restarted if the download for the current synchronization failed and can be restarted.

```
CREATE PROCEDURE sp_hook_dbmsync_end()
BEGIN
  -- Restart the download if the download for the current sync
  -- failed and can be restarted
  IF EXISTS (SELECT * FROM #hook_dict
    WHERE name = 'restartable download' AND value='true')
    THEN
      UPDATE #hook_dict SET value = 'download' WHERE name='restart';
    END IF;
END;
```

sp_hook_dbmsync_log_rescan

Use this stored procedure to programmatically decide when a rescan is required.

Rows in #hook_dict table

Name	Value	Description
publication_ <i>n</i> (in)	publication	Deprecated. Use subscription_ <i>n</i> instead. The publications being synchronized, where <i>n</i> is an integer. There is one publication_ <i>n</i> entry for each publication being synchronized. The numbering of <i>n</i> starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
discarded storage (in)	number	The number of bytes of discarded memory after the last synchronization.
rescan (in out)	true false	If set to True by the hook, dbmlsync performs a complete rescan before the next synchronization. On entry, this value is set to False.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_ <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

When more than one -n option or -s option is specified in the command line, dbmlsync may experience fragmentation which results in discarded memory. The discarded memory can be recovered by rescanning the database transaction log. This hook allows you to decide if dbmlsync should rescan the database transaction log to recover memory.

When no other condition has been met that would force a rescan, this hook is called immediately after the sp_hook_dbmlsync_process_exit_code hook.

See also

- [“HoverRescanThreshold \(hrt\) extended option” on page 135](#)

Examples

The following example cause a log scan if the discarded storage is greater than 1000 bytes.

```
CREATE PROCEDURE sp_hook_dbmlsync_log_rescan ()
BEGIN
  IF EXISTS(SELECT * FROM #hook_dict
    WHERE name = 'Discarded storage' AND value>1000)
  THEN
    UPDATE #hook_dict SET value = 'true' WHERE name='Rescan';
```



```

END IF;
END;

```

sp_hook_dbmsync_logscan_begin

Use this stored procedure to add custom actions immediately before the transaction log is scanned for upload.

Rows in #hook_dict table

Name	Value	Description
starting log offset_ <i>n</i> (in)	number	The progress value for each subscription being synchronized. The progress value is the offset in the transaction log up to which all data for the subscription has been uploaded. There is one value for each subscription being synchronized. The numbering of <i>n</i> starts at zero. This value matches subscription_ <i>n</i> . For example, log offset_0 is the offset for subscription_0.
log scan retry (in)	true false	If this is the first time the transaction log has been scanned for this synchronization, the value is false; otherwise it is true. The log is scanned twice when the MobiLink server and dbmsync have different information about where the scanning should begin.
publication_ <i>n</i> (in)	publication	Deprecated. Use subscription_ <i>n</i> instead. The publications being synchronized, where <i>n</i> is an integer. There is one publication_ <i>n</i> entry for each publication being synchronized. The numbering of <i>n</i> starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_ <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

If a procedure of this name exists, it is called immediately before dbmlsync scans the transaction log to assemble the upload.

This hook is ideal for making any last minute changes to the tables being synchronized that you want to include in the upload.

Actions of this procedure are committed immediately after execution.

See also

- [“Synchronization event hook sequence” on page 178](#)

Examples

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
  "event_id"          integer NOT NULL DEFAULT autoincrement ,
  "event_time"       timestamp NULL,
  "event_name"       varchar(128) NOT NULL ,
  "subs"             varchar(1024) NULL ,
  PRIMARY KEY ("event_id")
)
```

The following logs the beginning of the log scan for each synchronization.

```
CREATE PROCEDURE sp_hook_dbmlsync_logscan_begin ()
BEGIN

  DECLARE subs_list VARCHAR(1024);

  -- build a list of subscriptions being synchronized
  SELECT LIST(value) INTO subs_list
  FROM #hook_dict
  WHERE name LIKE 'subscription_%';

  -- log the event
  INSERT INTO SyncLog(event_time, event_name, subs)
  VALUES( CURRENT_TIMESTAMP, 'sp_hook_dbmlsync_logscan_begin', subs_list );
END
```

sp_hook_dbmlsync_logscan_end

Use this stored procedure to add custom actions immediately after the transaction log is scanned.

Rows in #hook_dict table

Name	Value	Description
ending log offset (in)	number	The log offset value where scanning ended.

Name	Value	Description
starting log offset_ <i>n</i> (in)	number	The initial progress value for each subscription you synchronize. The <i>n</i> values correspond to those in publication_ <i>n</i> . For example, Starting log offset_1 is the offset for publication_1.
log scan retry (in)	true false	If this is the first time the transaction log has been scanned for this synchronization, the value is false; otherwise it is true. The log is scanned twice when the MobiLink server and dbmlsync have different information about where the scanning should begin.
publication_ <i>n</i> (in)	publication	Deprecated. Use subscription_ <i>n</i> instead. The publications being synchronized, where <i>n</i> is an integer. There is one publication_ <i>n</i> entry for each publication being synchronized. The numbering of <i>n</i> starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_ <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

If a procedure of this name exists, it is called immediately after dbmlsync has scanned the transaction log.

Actions of this procedure are committed immediately after execution.

See also

- [“Synchronization event hook sequence” on page 178](#)

Examples

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
  "event_id"          integer NOT NULL DEFAULT autoincrement ,
  "event_time"       timestamp NULL,
  "event_name"       varchar(128) NOT NULL ,
```

```
"subs"          varchar(1024) NULL ,
  PRIMARY KEY ("event_id")
)
```

The following logs the end of log scanning for each synchronization.

```
CREATE PROCEDURE sp_hook_dbmlsync_logscan_end ()
BEGIN

  DECLARE subs_list VARCHAR(1024);

  -- build a list of subscriptions being synchronized
  SELECT LIST(value) INTO subs_list
  FROM #hook_dict
  WHERE name LIKE 'subscription_%';

  -- log the event
  INSERT INTO SyncLog(event_time, event_name, subs)
  VALUES( CURRENT_TIMESTAMP, 'sp_hook_dbmlsync_logscan_end', subs_list );
END
```

sp_hook_dbmlsync_misc_error

Use this stored procedure to process dbmlsync errors which are not categorized as database or communication errors. For example, you can implement the sp_hook_dbmlsync_misc_error hook to log errors or perform a specific action when a specific error occurs.

Rows in #hook_dict table

Name	Value	Description
publication_n (in)	publication	Deprecated. Use subscription_n instead. The publications being synchronized, where n is an integer. There is one publication_n entry for each publication being synchronized. The numbering of n starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
error message (in)	error message text	This is the same text that is displayed in the dbmlsync log.
error id (in)	integer	An ID that uniquely identifies the message. Use this row to identify the error message, as the error message text may change.

Name	Value	Description
error hook user state (in out)	integer	This value can be set by the hook to pass state information to future calls to the <code>sp_hook_dbmsync_all_error</code> , <code>sp_hook_dbmsync_communication_error</code> , <code>sp_hook_dbmsync_misc_error</code> , <code>sp_hook_dbmsync_sql_error</code> , or <code>sp_hook_dbmsync_end</code> hooks. The first time one of these hooks is called, the value of the row is 0. If a hook changes the value of the row, the new value is used in the next hook call.
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one <code>subscription_ <i>n</i></code> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

If an error occurs during startup before a synchronization has been initiated, the `#hook_dict` entries for MobiLink user and Script version are set to an empty string, and no `publication_ n` or `subscription_ n` rows are set in the `#hook_dict` table.

The error hook user state row provides a useful mechanism for you to pass information about the nature of the error to the `sp_hook_dbmsync_end` hook where you might use that information to decide whether to retry the synchronization.

This procedure executes on a separate connection to ensure that operations it performs are not lost if a rollback is performed on the synchronization connection. If `dbmsync` cannot establish a separate connection, the procedure is not called.

Since this hook executes on a separate connection you should use care when accessing tables that are being synchronized in your hook procedure because `dbmsync` may have locks on these tables. These locks could cause operations in your hook to fail or to wait indefinitely.

Actions of this procedure are committed immediately after execution.

See also

- [“Handling errors and warnings in event hook procedures” on page 182](#)
- [“sp_hook_dbmsync_communication_error” on page 189](#)
- [“sp_hook_dbmsync_all_error” on page 185](#)
- [“sp_hook_dbmsync_sql_error” on page 224](#)

Examples

Assume you use the following table to log errors in the remote database.

```
CREATE TABLE error_log
(
  pk INTEGER DEFAULT AUTOINCREMENT PRIMARY KEY,
  err_id INTEGER,
  err_msg VARCHAR(10240),
);
```

The following example sets up `sp_hook_dbmlsync_misc_error` to log all types of error messages.

```
CREATE PROCEDURE sp_hook_dbmlsync_misc_error()
BEGIN
  DECLARE msg VARCHAR(10240);
  DECLARE id INTEGER;

  // get the error message text
  SELECT value INTO msg
  FROM #hook_dict
  WHERE name = 'error message';

  // get the error id
  SELECT value INTO id
  FROM #hook_dict
  WHERE name = 'error id';

  // log the error information
  INSERT INTO error_log(err_msg,err_id)
  VALUES (msg,id);
END;
```

To see possible error id values, test run `dbmlsync`. For example, the following `dbmlsync` command line references an invalid subscription.

```
dbmlsync -c server=custdb;uid=DBA;pwd=sql -s test
```

Now, the `error_log` table contains the following row, associating the error with the error id 9931.

```
1,19912,
'Subscription ''test'' not found.'
```

To provide custom error handling, check for the error id 19912 in `sp_hook_dbmlsync_misc_error`.

```
ALTER PROCEDURE sp_hook_dbmlsync_misc_error()
BEGIN
  DECLARE msg VARCHAR(10240);
  DECLARE id INTEGER;

  // get the error message text
  SELECT value INTO msg
  FROM #hook_dict
  WHERE name = 'error message';

  // get the error id
  SELECT value INTO id
  FROM #hook_dict
  WHERE name = 'error id';

  // log the error information
  INSERT INTO error_log(err_msg,err_id)
  VALUES (msg,id);

  IF id = 19912 THEN
    // handle invalid subscription
```

```

END IF;

END;

```

sp_hook_dbmlsync_ml_connect_failed

Use this stored procedure to retry failed attempts to connect to the MobiLink server using a different communication type or address.

Rows in #hook_dict table

Name	Value	Description
publication_ <i>n</i> (in)	publication	Deprecated. Use subscription_ <i>n</i> instead. The publications being synchronized, where <i>n</i> is an integer. There is one publication_ <i>n</i> entry for each publication being synchronized. The numbering of <i>n</i> starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
connection address (in/out)	connection address	When the hook is invoked, this is the address used in the most recent failed attempt to connect. You can set this value to a new connection address that you want to try. If retry is set to true, this value is used for the next attempt to connect. For a list of protocol options, see “MobiLink client network protocol option summary” on page 23 .
connection type (in/out)	network protocol	When the hook is invoked, this is the network protocol (such as TCPIP) that was used in the most recent failed attempt to connect. You can set this value to a new network protocol that you want to try. If retry is set to true, this value is used for the next attempt to connect. For a list of network protocols, see “CommunicationType (ctp) extended option” on page 129 .

Name	Value	Description
user data (in out)	user-defined data	State information to be used if the next connection attempt fails. For example, you might find it useful to store the number of retries that have occurred. The default is an empty string.
allow remote ahead (in out)	true false	This is true only if the dbmsync -ra option or the RemoteProgressGreater=on synchronization profile option was specified for this synchronization. By changing the value of this row, you can change the value of the option for the current synchronization only. See “ -r dbmsync option ” on page 116.
allow remote behind (in out)	true false	This is true only if the dbmsync -ra option or the RemoteProgressLess=on synchronization profile option was specified for this synchronization. By changing the value of this row, you can change the value of the option for the current synchronization only. See “ -r dbmsync option ” on page 116.
retry (in out)	true false	Set this value to true if you want to retry a failed connection attempt. The default is FALSE.
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_ <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

If a procedure of this name exists, it is called if dbmsync fails to connect to the MobiLink server.

This hook only applies to connection attempts to the MobiLink server, not the database.

When a progress offset mismatch occurs, dbmsync disconnects from the MobiLink server and reconnects later. In this kind of reconnection, this hook is not called, and failure to reconnect causes the synchronization to fail.

Actions of this procedure are committed immediately after execution.

Examples

This example uses the sp_hook_dbmsync_ml_connect_failed hook to retry the connection up to five times.


```

CREATE PROCEDURE sp_hook_dbmlsync_ml_connect_failed ()
BEGIN
    DECLARE idx integer;

    SELECT IF value = ''then 0 else cast(value as integer)endif
    INTO idx
    FROM #hook_dict
    WHERE name = 'user data';

    IF idx < 5 THEN
        UPDATE #hook_dict
        SET value = idx +1
        WHERE name = 'user data';

        UPDATE #hook_dict
        SET value = 'TRUE'
        WHERE name = 'retry';
    END IF;
END;

```

The next example uses a table containing connection information. When an attempt to connect fails, the hook tries the next server in the list.

```

CREATE TABLE conn_list (
    label    INTEGER PRIMARY KEY,
    addr    VARCHAR( 128 ),
    type    VARCHAR( 64 )
);
INSERT INTO conn_list
VALUES ( 1, 'host=server1;port=91', 'tcpip' );
INSERT INTO conn_list
VALUES ( 2, 'host=server2;port=92', 'http' );
INSERT INTO conn_list
VALUES ( 3, 'host=server3;port=93', 'tcpip' );
COMMIT;

CREATE PROCEDURE sp_hook_dbmlsync_ml_connect_failed ()
BEGIN
    DECLARE idx INTEGER;
    DECLARE cnt INTEGER;

    SELECT if value = ''then | else cast(value as integer)endif
    INTO idx
    FROM #hook_dict
    WHERE name = 'user data';

    SELECT COUNT( label ) INTO cnt FROM conn_list;

    IF idx <= cnt THEN
        UPDATE #hook_dict
        SET value = ( SELECT addr FROM conn_list WHERE label = idx )
        WHERE name = 'connection address';
        UPDATE #hook_dict
        SET value = (SELECT type FROM conn_list WHERE label=idx)
        WHERE name = 'connection type';

        UPDATE #hook_dict
        SET value = idx +1
        WHERE name = 'user data';

        UPDATE #hook_dict
        SET value = 'TRUE'
        WHERE name = 'retry';
    END IF;
END;

```

```

END IF;
END;

```

sp_hook_dbmsync_process_exit_code

Use this stored procedure to manage exit codes.

Rows in #hook_dict table

Name	Value	Description
publication_ <i>n</i> (in)	publication	Deprecated. Use subscription_ <i>n</i> instead. The publications being synchronized, where <i>n</i> is an integer. There is one publication_ <i>n</i> entry for each publication being synchronized. The numbering of <i>n</i> starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
fatal error (in)	true false	True when the hook is called because of an error that causes dbmsync to terminate.
aborted synchronization (in)	true false	True when the hook is called because of an abort request from the sp_hook_dbmsync_abort hook.
exit code (in)	number	The exit code from the most recent synchronization attempt. 0 indicates a successful synchronization. Any other value indicates that the synchronization failed. This value can be set by sp_hook_dbmsync_abort when that hook is used to abort synchronization.
last exit code (in)	number	The value stored in the new exit code row of the #hook_dict table the last time this hook was called, or 0 if this is the first call to the hook.
new exit code (in out)	number	The exit code you choose for the process. When dbmsync exits, its exit code is the value stored in this row by the last call to the hook. The value must be -32768 to 32767.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.

Name	Value	Description
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_ <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

A dbmlsync session can run multiple synchronizations when you specify the -n option or -s option more than once in the command line, when you use scheduling, or when you use the restart parameter in sp_hook_dbmlsync_end. In these cases, if one or more of the synchronizations fail, the default exit code does not indicate which failed. Use this hook to define the exit code for the dbmlsync process based on the exit codes from the synchronizations. This hook can also be used to log exit codes.

If an error occurs during startup before a synchronization has been initiated, the #hook_dict entries for MobiLink user and Script version are set to an empty string, and no publication_*n* or subscription_*n* rows are set in the #hook_dict table.

Example

Suppose that you run dbmlsync to perform five synchronizations and you want the exit code to indicate how many of the synchronizations failed, with an exit code of 0 indicating that there were no failures, an exit code of 1 indicating that one synchronization failed, and so on. You can achieve this by defining the sp_hook_dbmlsync_process_exit_code hook as follows. In this case, if three synchronizations fail, the new exit code is 3.

```
CREATE PROCEDURE sp_hook_dbmlsync_process_exit_code()
BEGIN
    DECLARE rc INTEGER;

    SELECT value INTO rc FROM #hook_dict WHERE name = 'exit code';
    IF rc <> 0 THEN
        SELECT value INTO rc FROM #hook_dict WHERE name = 'last exit code';
        UPDATE #hook_dict SET value = rc + 1 WHERE name = 'new exit code';
    END IF;
END;
```

See also

- [“Synchronization event hook sequence” on page 178](#)
- [“sp_hook_dbmlsync_abort” on page 183](#)

sp_hook_dbmlsync_schema_upgrade

Use this stored procedure to run a SQL script that revises your schema.

Rows in #hook_dict table

Name	Value	Description
publication_ <i>n</i> (in)	publication	Deprecated. Use subscription_ <i>n</i> instead. The publications being synchronized, where <i>n</i> is an integer. There is one publication_ <i>n</i> entry for each publication being synchronized. The numbering of <i>n</i> starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	name of script version	The script version used for the synchronization.
drop hook (out)	never always on success	The values can be: never - (the default) Do not drop the sp_hook_dbmlsync_schema_upgrade hook from the database. always - After attempting to run the hook, drop the sp_hook_dbmlsync_schema_upgrade hook from the database. on success - If the hook runs successfully, drop the sp_hook_dbmlsync_schema_upgrade hook from the database. On success is identical to always if the dbmlsync -eh option is used, or the dbmlsync extended option IgnoreHookErrors is set to true, or the IgnoreHookErrors synchronization profile option is set to on.
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_ <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

This hook is primarily provided for backward compatibility purposes. Unless you are using the ScriptVersion extended option you can safely perform schema changes without using this hook by using the START SYNCHRONIZATION SCHEMA CHANGE statement. See [“START SYNCHRONIZATION SCHEMA CHANGE statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#).

When this hook is implemented, dbmlsync locks the tables being synchronized by default.

This stored procedure is intended for making schema changes to deployed remote databases. Using this hook for schema upgrades ensures that all changes on the remote database are synchronized before the

schema is upgraded, which ensures that the database continues to synchronize. When this hook is being used you should not set the dbmsync extended option LockTables to off.

During any synchronization where the upload was applied successfully and acknowledged by MobiLink, this hook is called after the sp_hook_dbmsync_download_end hook and before the sp_hook_dbmsync_end hook. This hook is not called during download-only synchronization or when a file-based download is being created or applied.

Actions performed in this hook are committed immediately after the hook completes. It is safe to commit or rollback in this hook.

See also

- [“Schema changes in remote clients” on page 58](#)

Examples

The following example uses the sp_hook_dbmsync_schema_upgrade procedure to add a column to the Dealer table on the remote database. If the upgrade is successful the sp_hook_dbmsync_schema_upgrade hook is dropped.

```
CREATE PROCEDURE sp_hook_dbmsync_schema_upgrade()
BEGIN
  -- Upgrade the schema of the Dealer table. Add a column:
  ALTER TABLE Dealer
  ADD dealer_description VARCHAR(128);

  -- Change the script version used to synchronize
  ALTER SYNCHRONIZATION SUBSCRIPTION sub1
  SET SCRIPT VERSION='v2';

  -- If the schema upgrade is successful, drop this hook:
  UPDATE #hook_dict
  SET value = 'on success'
  WHERE name = 'drop hook';
END;
```

sp_hook_dbmsync_set_extended_options

Use this stored procedure to programmatically customize the behavior of an upcoming synchronization by specifying extended options to be applied to that synchronization.

Rows in #hook_dict table

Name	Value	Description
publication_n (in)	publication	Deprecated. Use subscription_n instead. The publications being synchronized, where <i>n</i> is an integer. There is one publication_n entry for each publication being synchronized. The numbering of <i>n</i> starts at zero.

Name	Value	Description
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
extended options (out)	<i>opt=val;...</i>	Extended options to add for the next synchronization.
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_ <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

If a procedure of this name exists, it is called one or more times before each synchronization.

Extended options specified by this hook apply only to the synchronization identified by the subscription and MobiLink user entries, and they apply only until the next time the hook is called for the same synchronization.

The Schedule extended option may not be specified using this hook.

Actions of this procedure are committed immediately after execution.

See also

- “Synchronization event hook sequence” on page 178
- “MobiLink SQL Anywhere client extended options” on page 126
- “Priority order” on page 126

Examples

The following example uses `sp_hook_dbmlsync_set_extended_options` to specify the `SendColumnNames` extended option. The extended option is only applied if `sub1` is synchronizing.

```
CREATE PROCEDURE sp_hook_dbmlsync_set_extended_options ()
BEGIN
  IF exists(SELECT * FROM #hook_dict
    WHERE name LIKE 'subscription_%' AND value='sub1')
  THEN
    -- specify the SendColumnNames=on extended option
    UPDATE #hook_dict
      SET value = 'SendColumnNames=on'
      WHERE name = 'extended options';
  END IF;
END;
```

sp_hook_dbmlsync_set_ml_connect_info

Use this stored procedure to set the network protocol and network protocol option used to connect to the MobiLink server.

Name	Value	Description
publication_ <i>n</i> (in)	publication	Deprecated. Use subscription_ <i>n</i> instead. The publications being synchronized, where <i>n</i> is an integer. There is one publication_ <i>n</i> entry for each publication being synchronized. The numbering of <i>n</i> starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
connection type (in/out)	tcpip, tls, http, or https	The network protocol that is used to connect to the MobiLink server.
connection address (in/out)	protocol options	The communication address that is used to connect to the MobiLink server. See “MobiLink client network protocol option summary” on page 23.
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_ <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

You can use this hook to set the network protocol and network protocol options used to connect to the MobiLink server by changing the value in the connection type and/or connection address rows.

The protocol and options can also be set in the `sp_hook_dbmsync_set_extended_options`, a hook that is called at the beginning of a synchronization. `sp_hook_dbmsync_set_ml_connect_info` is called immediately before `dbmsync` attempts to connect to the MobiLink server.

This hook is useful when you want to set options in a hook, but want to do so later in the synchronization process than the `sp_hook_dbmsync_set_extended_options`. For example, if the options should be set based on the availability of signal strength of the network that is being used.

See also

- [“Introduction to dbmsync hooks”](#) on page 178
- [“Synchronization event hook sequence”](#) on page 178
- [“CommunicationType \(ctp\) extended option”](#) on page 129
- [“MobiLink client network protocol option summary”](#) on page 23
- [“sp_hook_dbmsync_set_extended_options”](#) on page 219

Example

```
CREATE PROCEDURE sp_hook_dbmlsync_set_ml_connect_info()
begin
  UPDATE #hook_dict
  SET VALUE = 'tcpip'
  WHERE name = 'connection type';

  UPDATE #hook_dict
  SET VALUE = 'host=localhost'
  WHERE name = 'connection address';
end
```

sp_hook_dbmlsync_set_upload_end_progress

This stored procedure can be used to define an ending progress when a scripted upload subscription is synchronized. This procedure is called only when a scripted upload publication is being synchronized.

Rows in #hook_dict table

Name	Value	Description
generating download exclusion list (in)	TRUE FALSE	TRUE if no upload will be sent during the synchronization (for example, in a download-only synchronization or when a file-based download is applied). In these cases, the upload scripts are still called and the operations generated are used to identify download operations that change rows that need to be uploaded. When such an operation is found, the download is not applied.
publication_n (in)	publication	Deprecated. Use subscription_n instead. The publications being synchronized, where n is an integer. There is one publication_n entry for each publication being synchronized. The numbering of n starts at zero.
start progress as timestamp_n	progress as timestamp	The starting progress for each subscription being synchronized expressed as a timestamp, where n is the same integer used to identify the subscription.
start progress as bigint_n	progress as bigint	The starting progress for each subscription being synchronized expressed as a bigint, where n is the same integer used to identify the subscription.
script version (n)	script version name	The MobiLink script version to be used for the synchronization.

Name	Value	Description
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
end progress is bigint (in out)	TRUE FALSE	<p>When this row is set to TRUE, the end progress value is assumed to be an unsigned bigint that is represented as a string (for example, '12345').</p> <p>When this row is set to FALSE, the end progress value is assumed to be a timestamp that is represented as a string (for example, '1900/01/01 12:00:00.000').</p> <p>The default is FALSE.</p>
end progress (in out)	timestamp	<p>The hook can modify this row to change the "end progress as bigint" and "end progress as timestamp" values passed to the upload scripts. These values define the point in time up to which all operations are included in the upload that is being generated.</p> <p>The value of this row can be set as either an unsigned bigint or as a timestamp according to the setting of the "progress is bigint" row. The default value for this row is the current timestamp.</p>
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_ <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

For a scripted upload, each time an upload procedure is called it is passed a start progress value and an end progress value. The procedure must return all appropriate operations that occurred during the period defined by those two values. The begin progress value is always the same as the end progress value from the last successful synchronization, unless this is a first synchronization, in which case the begin progress is January 1, 1900, 00:00:00.000. By default, the end progress value is the time when dbmlsync began building the upload.

This hook lets you override the default end progress value. You could define a shorter period for the upload or you could implement a progress tracking scheme based on something other than timestamps (for example, generation numbers).

If "end progress is bigint" is set to true, the end progress must be an integer less than or equal to the number of milliseconds from 1900-01-01 00:00:00 to 9999-12-31 23:59:59:999, which is 255,611,203,259,999.

See also

- [“Custom progress values in scripted upload” on page 319](#)
- [“Synchronization event hook sequence” on page 178](#)
- [“Scripted upload” on page 310](#)

sp_hook_dbmsync_sql_error

Use this stored procedure to handle database errors that occur during synchronization. For example, you can implement the `sp_hook_dbmsync_sql_error` hook to perform a specific action when a specific SQL error occurs.

Rows in #hook_dict table

Name	Value	Description
publication_ <i>n</i> (in)	publication	Deprecated. Use subscription_ <i>n</i> instead. The publications being synchronized, where <i>n</i> is an integer. There is one publication_ <i>n</i> entry for each publication being synchronized. The numbering of <i>n</i> starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
error message (in)	error message text	This is the same text that is displayed in the dbmsync log.
error id (in)	numeric	An ID that uniquely identifies the message.
error hook user state (in out)	integer	This value can be set by the hook to pass state information to future calls to the <code>sp_hook_dbmsync_all_error</code> , <code>sp_hook_dbmsync_communication_error</code> , <code>sp_hook_dbmsync_misc_error</code> , <code>sp_hook_dbmsync_sql_error</code> , or <code>sp_hook_dbmsync_end</code> hooks. The first time one of these hooks is called, the value of the row is 0. If a hook changes the value of the row, the new value is used in the next hook call.

Name	Value	Description
SQLCODE (in)	SQLCODE	The SQLCODE returned by the database when the operation failed. See “SQL Anywhere error messages sorted by SQLCODE” [Error Messages] .
SQLSTATE (in)	SQLSTATE value	The SQLSTATE returned by the database when the operation failed.
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_ <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

If an error occurs during startup before a synchronization has been initiated, the #hook_dict entries for MobiLink user and Script version are set to an empty string, and no publication_*n* or subscription_*n* rows are set in the #hook_dict table.

You can identify SQL errors using the SQL Anywhere SQLCODE or the ANSI SQL standard SQLSTATE. For a list of SQLCODE or SQLSTATE values, see [“SQL Anywhere error messages” \[Error Messages\]](#).

The error hook user state row provides a useful mechanism for you to pass information about the nature of the error to the sp_hook_dbmsync_end hook where you might use that information to decide whether to retry the synchronization.

This procedure executes on a separate connection to ensure that operations it performs are not lost if a rollback is performed on the synchronization connection. If dbmsync cannot establish a separate connection, the procedure is not called.

Since this hook executes on a separate connection you should use care when accessing tables that are being synchronized in your hook procedure because dbmsync may have locks on these tables. These locks could cause operations in your hook to fail or to wait indefinitely.

Actions of this procedure are committed immediately after execution.

See also

- [“Handling errors and warnings in event hook procedures” on page 182](#)
- [“sp_hook_dbmsync_all_error” on page 185](#)
- [“sp_hook_dbmsync_communication_error” on page 189](#)
- [“sp_hook_dbmsync_misc_error” on page 210](#)
- [“SQL Anywhere error messages” \[Error Messages\]](#)

sp_hook_dbmsync_upload_begin

Use this stored procedure to add custom actions immediately before the transmission of the upload.

Rows in #hook_dict table

Name	Value	Description
Publication_ <i>n</i> (in)	publication	Deprecated. Use subscription_ <i>n</i> instead. The publications being synchronized, where <i>n</i> is an integer. There is one publication_ <i>n</i> entry for each publication being synchronized. The numbering of <i>n</i> starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
Script version (in)	script version name	The MobiLink script version to be used for the synchronization.
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_ <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

If a procedure of this name exists, it is called immediately before dbmsync sends the upload.

Actions of this procedure are committed immediately after execution.

See also

- [“Synchronization event hook sequence” on page 178](#)

Examples

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
  "event_id"          integer NOT NULL DEFAULT autoincrement ,
  "event_time"       timestamp NULL,
  "event_name"       varchar(128) NOT NULL ,
  "subs"             varchar(1024) NULL ,
  PRIMARY KEY ("event_id")
)
```

The following logs the beginning of the upload for each synchronization.

```
CREATE PROCEDURE sp_hook_dbmsync_upload_begin ()
BEGIN
  DECLARE subs_list VARCHAR(1024);
```

```

-- build a list of subscriptions being synchronized
SELECT LIST(value) INTO subs_list
FROM #hook_dict
WHERE name LIKE 'subscription_%';

-- log the event
INSERT INTO SyncLog(event_time, event_name, subs)
VALUES( CURRENT_TIMESTAMP, 'sp_hook_dbmsync_upload_begin', subs_list );
END

```

sp_hook_dbmsync_upload_end

Use this stored procedure to add custom actions after dbmsync has verified receipt of the upload by the MobiLink server.

Rows in #hook_dict table

Name	Value	Description
failure cause (in)	See range of values in Remarks, below	The cause of failure of an upload. For more information, see Remarks.
upload status (in)	retry committed failed unknown	<p>Specifies the status returned by the MobiLink server when dbmsync attempted to verify receipt of the upload.</p> <p>retry - The MobiLink server and dbmsync had different values for the log offset from which the upload should start. The upload was not committed by the MobiLink server. The dbmsync utility attempts to send another upload starting from a new log offset.</p> <p>committed - The upload was received by the MobiLink server and committed.</p> <p>failed - The MobiLink server did not commit the upload.</p> <p>unknown - The upload was not acknowledged by the MobiLink server. There is no way to know if it was committed or not.</p>
publication_n (in)	publication	Deprecated. Use subscription_n instead. The publications being synchronized, where n is an integer. There is one publication_n entry for each publication being synchronized. The numbering of n starts at zero.

Name	Value	Description
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
authentication value (in)	value	This value indicates the results of dbmlsync's attempt to authenticate to the MobiLink server. It is generated by the <code>authenticate_user</code> , <code>authenticate_user_hashed</code> , or <code>authenticate_parameters</code> script on the server. The value is an empty string when the upload status is unknown or when the <code>upload_end</code> hook is called after an upload is resent because of a conflict between the log offsets stored in the remote and consolidated databases.
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one <code>subscription_</code> <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

If a procedure of this name exists, it is called immediately after dbmlsync has sent the upload and received confirmation of it from the MobiLink server.

When performing a transactional upload or an incremental upload this hook is called after each segment of the upload is sent. In these cases, the upload status will be "unknown" each time the hook is called except for the last time.

Actions of this procedure are committed immediately after execution.

The range of possible parameter values for the failure cause row in the `#hook_dict` table includes:

- **UPLD_ERR_INVALID_USERID_OR_PASSWORD** The user ID or password was incorrect.
- **UPLD_ERR_USERID_OR_PASSWORD_EXPIRED** The user ID or password expired.
- **UPLD_ERR_REMOTE_ID_ALREADY_IN_USE** The remote ID was already in use.
- **UPLD_ERR_SQLCODE_***n* Here, *n* is an integer. A SQL error occurred in the consolidated database. The integer specified is the SQLCODE for the error encountered.
- **UPLD_ERR_USER_ABORT_REQUEST** The upload was aborted at the user's request.

See also

- [“Synchronization event hook sequence” on page 178](#)

Examples

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
  "event_id"          integer NOT NULL DEFAULT autoincrement ,
  "event_time"       timestamp NULL,
  "event_name"       varchar(128) NOT NULL ,
  "subs"             varchar(1024) NULL ,
  PRIMARY KEY ("event_id")
)
```

The following logs the end up the upload for each synchronization.

```
CREATE PROCEDURE sp_hook_dbmlsync_upload_end ()
BEGIN

  DECLARE subs_list VARCHAR(1024);

  -- build a list of subscriptions being synchronized
  SELECT LIST(value) INTO subs_list
  FROM #hook_dict
  WHERE name LIKE 'subscription_%';

  -- log the event
  INSERT INTO SyncLog(event_time, event_name, subs)
  VALUES( CURRENT_TIMESTAMP, 'sp_hook_dbmlsync_upload_end', subs_list );
END
```

sp_hook_dbmlsync_validate_download_file

Use this hook to implement custom logic to decide if a download file can be applied to the remote database. This hook is called only when a file-based download is applied.

Rows in #hook_dict table

Name	Value	Description
publication_ <i>n</i> (in)	publication	Deprecated. Use subscription_ <i>n</i> instead. The publications being synchronized, where <i>n</i> is an integer. There is one publication_ <i>n</i> entry for each publication being synchronized. The <i>n</i> in publication_ <i>n</i> and generation number_ <i>n</i> match. The numbering of <i>n</i> starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.

Name	Value	Description
file last download time (in)		The download file's last download time. (The download file contains all rows that were changed between its last download time and its next last download time.)
file next last download time (in)		The download file's next last download time. (The download file contains all rows that were changed between its last download time and its next last download time.)
file creation time (in)		The time when the download file was created.
file generation number_ <i>n</i> (in)	number	The generation numbers from the download file. There is one file generation number_ <i>n</i> for each subscription_ <i>n</i> entry. The <i>n</i> in subscription_ <i>n</i> and generation number_ <i>n</i> match. The numbering of <i>n</i> starts at zero.
user data (in)	string	The string specified with the dbmlsync -be option when the download file was created or the DnldFileExtra synchronization profile option.
apply file (in out)	True False	If true (the default), the download file is applied only if it passes dbmlsync's other validation checks. If false, the download file is not applied to the remote database.
check generation number (in out)	True False	If true (the default), dbmlsync validates generation numbers. If the generation numbers in the download file do not match those in the remote database, dbmlsync does not apply the download file. If false, dbmlsync does not check generation numbers.
setting generation number (in)	true false	True if the -bg option or UpdateGenNum synchronization profile option was used when the download file was created. When true, the generation numbers on the remote database are updated from the download file and normal generation number checks are not performed.
subscription_ <i>n</i> (in)	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_ <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.

Remarks

Use this stored procedure to implement custom checks to decide if a download file can be applied.

If you want to compare the generation numbers or timestamps contained in the file with those stored in the remote database, they can be queried from the SYSSYNC system view.

This hook before the a file-based download is applied to the remote database.

The actions of this hook are committed immediately after it completes.

See also

- “-be dbmsync option” on page 99
- “-bg dbmsync option” on page 99
- “MobiLink file-based download” [*MobiLink - Server Administration*]

Examples

The following example prevents application of download files that don't contain the user string 'sales manager data'.

```
CREATE PROCEDURE sp_hook_dbmsync_validate_download_file ()
BEGIN
  IF NOT exists(SELECT * FROM #hook_dict
    WHERE name = 'User data' AND value='sales manager data')
  THEN
    UPDATE #hook_dict
      SET value = 'false' WHERE name = 'Apply file';
  END IF;
END;
```

Dbmsync C++ API reference

Header file

dbmsynccli.hpp

Example

The sample below shows a typical application using the C++ version of the Dbmsync API to perform a synchronization and receive output events. The sample omits error handling for clarity. It is always good practice to check the return value from each API call.

```
#include <stdio.h>
#include "dbmsynccli.hpp"

int main( void ) {
  DbmsyncClient *client;
  DBSC_SyncHdl   syncHdl;
  DBSC_Event     *ev1;

  client = DbmsyncClient::InstantiateClient();
  if( client == NULL ) return( 1 );
  client->Init();
```

```

// Setting the "server path" is usually required on Windows Mobile/CE.
// In other environments the server path is usually not required unless
// you SA install is not in your path or you have multiple versions of
// the product installed
client->SetProperty( "server path", "C:\\SQLAnywhere\\bin32" );

client->StartServer( 3426,
    "-c server=remote;dbn=reml;uid=dba;pwd=sql -v+ -ot c:\\
\dbsync1.txt",
    5000, NULL );
client->Connect( NULL, 3426, "dba", "sql");
syncHdl = client->Sync( "my_sync_profile", "" );
while( client->GetEvent( &ev1, 5000) == DBSC_GETEVENT_OK ) {
    if( ev1->hdl == syncHdl ) {
        //
        // Process events that interest you here
        //

        if( ev1->type == DBSC_EVENTTYPE_SYNC_DONE ) {
            client->FreeEventInfo( ev1 );
            break;
        }
        client->FreeEventInfo( ev1 );
    }
}
client->ShutdownServer( DBSC_SHUTDOWN_ON_EMPTY_QUEUE );
client->WaitForServerShutdown( 10000 );
client->Disconnect();
client->Fini();
delete client;

return( 0 );
}

```

DbmlsyncClient class

Communicates using TCP/IP with a separate process, dbmlsync server, which performs a synchronization by connecting to the MobiLink server and the remote database.

Syntax

```
public class DbmlsyncClient
```

Members

All members of DbmlsyncClient class, including all inherited members.

Name	Description
"CancelSync method"	Cancels a synchronization request.
"Connect method"	Opens a connection to a Dbmlsync server that is already running on this computer.

Name	Description
“Disconnect method”	Breaks the Dbmsync server connection that was established with the Connect method.
“Fini method”	Frees all resources used by this class instance.
“FreeEventInfo method”	Frees memory associated with a DBSC_Event structure returned by the GetEvent method.
“GetErrorInfo method”	Retrieves additional information about the failure after a DbmsyncClient class method returns a failed return code.
“GetEvent method”	Retrieves the next feedback event for synchronizations requested by the client.
“GetProperty method”	Retrieves the current value of a property.
“Init method”	Initializes a DbmsyncClient class instance.
“InstantiateClient method”	Creates an instance of the dbmsync client class that can be used to control synchronizations.
“Ping method”	Sends a ping request to the dbmsync server to check if the server is active and responding to requests.
“SetProperty method”	Sets various properties to modify the behavior of the class instance.
“ShutdownServer method”	Shuts down the dbmsync server to which the client is connected.
“StartServer method”	Starts a new dbmsync server if one is not already listening on the specified port.
“Sync method”	Requests that the dbmsync server perform a synchronization.
“WaitForServerShutdown method”	Returns when the server has shutdown or when the timeout expires, whichever comes first.

Remarks

Multiple clients can share the same dbmsync server. However, each dbmsync server can only synchronize a single remote database. Each remote database can have only one dbmsync server synchronizing it.

The dbmsync server performs one synchronization at a time. If the server receives a synchronization request while performing a synchronization, it queues that request and satisfies it later.

Status information generated by synchronizations is communicated back to the client application through the GetEvent method.

See also

- [“GetEvent method” on page 238](#)

CancelSync method

Cancels a synchronization request.

Overload list

Name	Description
“CancelSync(DBSC_SyncHdl) method (deprecated)”	Allows a client to cancel a synchronization request previously made using the Sync method.
“CancelSync(DBSC_SyncHdl, bool) method”	Allows a client to cancels a synchronization request previously made using the Sync method.

CancelSync(DBSC_SyncHdl) method (deprecated)

Allows a client to cancel a synchronization request previously made using the Sync method.

Syntax

```
public virtual bool CancelSync(DBSC_SyncHdl hdl)
```

Parameters

- **hdl** The synchronization handle returned by the Sync method when the synchronization was requested.

Returns

True when the synchronization request was successfully canceled; otherwise, returns false. When false is returned, you can call the GetErrorInfo method for more information about the failure.

Remarks

Only synchronization requests waiting to be serviced can be canceled. To stop a synchronization that has already begun, use the CancelSync(UInt32, Boolean) method.

You can use the ShutdownServer method and pass the DBSC_SHUTDOWN_CLEANLY type to cancel an active synchronization. Dbmlsync attempts to cancel the synchronization before shutting down. This task is equivalent of cancelling a synchronization using the DBTools interface.

A connection must be established to the server before this method can be used. This method cannot be used if the client has disconnected from the server since the Sync method was called.

See also

- [“GetErrorInfo method” on page 238](#)
- [“CancelSync method” on page 234](#)
- [“ShutdownServer method” on page 242](#)

CancelSync(DBSC_SyncHdl, bool) method

Allows a client to cancel a synchronization request previously made using the Sync method.

Syntax

```
public virtual DBSC_CancelRet CancelSync(
    DBSC_SyncHdl hdl,
    bool cancel_active
)
```

Parameters

- **hdl** The synchronization handle returned by the Sync method when the synchronization was requested.
- **cancel_active** When set to true, the request is cancelled even if the synchronization has already begun. When set to false, the request is only cancelled if synchronization has not begun.

Returns

A value from the DBSC_CancelRet enumeration. When DBSC_CANCEL_FAILED is returned, you can call the GetErrorInfo method for more information about the failure.

Remarks

You can use the ShutdownServer method and pass the DBSC_SHUTDOWN_CLEANLY type to cancel an active synchronization. Dbmsync attempts to cancel the synchronization before shutting down. This task is equivalent of cancelling a synchronization using the DBTools interface.

A connection must be established to the server before this method can be used. This method cannot be used if the client has disconnected from the server since the Sync method was called.

See also

- [“DBSC_CancelRet enumeration” on page 245](#)
- [“GetErrorInfo method” on page 238](#)
- [“ShutdownServer method” on page 242](#)

Connect method

Opens a connection to a Dbmsync server that is already running on this computer.

Syntax

```
public virtual bool Connect(
    const char * host,
    unsigned port,
```

```
    const char * uid,  
    const char * pwd  
)
```

Parameters

- **host** This value is reserved. Use NULL.
- **port** The TCP port on which the dbmlsync server is listening. Use the same port value that you specified with the StartServer method.
- **uid** A valid database user id with DBA or REMOTE DBA authority on the remote database that is to be synchronized.
- **pwd** The database password for the user specified by uid.

Returns

True when a connection to the server was established; otherwise, returns false. When false is returned, you can call the GetErrorInfo method for more information about the failure.

Remarks

The database user id and password are used to validate whether this client has enough permissions to synchronize the database. When synchronizations are performed, the user id that was specified with the -c option when the dbmlsync server started is used.

See also

- [“StartServer method” on page 243](#)
- [“GetErrorInfo method” on page 238](#)

Disconnect method

Breaks the Dbmlsync server connection that was established with the Connect method.

Syntax

```
public virtual bool Disconnect(void )
```

Returns

True when the connection to the server has been broken; otherwise, returns false. When false is returned, you can call the GetErrorInfo method for more information about the failure.

Remarks

You should always call Disconnect when you are finished with a connection.

See also

- [“Connect method” on page 235](#)
- [“GetErrorInfo method” on page 238](#)

Fini method

Frees all resources used by this class instance.

Syntax

```
public virtual bool Fini(void )
```

Returns

True when the class instance is successfully finalized; otherwise, returns false. When false is returned, you can call the `GetErrorInfo` method for more information about the failure.

Remarks

This method must be called before you can delete the `DbmlSyncClient` class instance.

Note

You should use the `Disconnect` method to disconnect from any connected servers before finalizing the class instance.

See also

- [“Disconnect method” on page 236](#)
- [“GetErrorInfo method” on page 238](#)

FreeEventInfo method

Frees memory associated with a `DBSC_Event` structure returned by the `GetEvent` method.

Syntax

```
public virtual bool FreeEventInfo(DBSC_Event * event)
```

Parameters

- **event** A pointer to the `DBSC_Event` structure to be freed.

Returns

True when the memory was successfully freed; otherwise, returns false. When false is returned, you can call the `GetErrorInfo` method for more information about the failure.

Remarks

`FreeEventInfo` must be called on each `DBSC_Event` structure returned by the `GetEvent` method.

See also

- [“DBSC_Event structure” on page 252](#)
- [“GetEvent method” on page 238](#)
- [“GetErrorInfo method” on page 238](#)

GetErrorInfo method

Retrieves additional information about the failure after a DbmlsyncClient class method returns a failed return code.

Syntax

```
public virtual const DBSC_ErrorInfo * GetErrorInfo(void )
```

Returns

A pointer to a DBSC_ErrorInfo structure that contains information about the failure. The contents of this structure may be overwritten the next time any class method is called.

See also

- [“DBSC_ErrorType enumeration” on page 246](#)
- [“DBSC_ErrorInfo structure” on page 251](#)
- [“GetErrorInfo method” on page 238](#)

GetEvent method

Retrieves the next feedback event for synchronizations requested by the client.

Syntax

```
public virtual DBSC_GetEventRet GetEvent(  
    DBSC_Event ** event,  
    unsigned timeout  
)
```

Parameters

- **event** If the return value is DBSC_GETEVENT_OK then the event parameter is filled in with a pointer to a DBSC_Event structure containing information about the event that has been retrieved. When you are finished with the event structure you must call the FreeEventInfo method to free memory associated with it.
- **timeout** Indicates the maximum time in milliseconds to wait if no event is immediately available to return. Use DBSC_INFINITY to wait indefinitely for a response.

Returns

A value from the DBSC_GetEventRet enumeration. When DBSC_GETEVENT_FAILED is returned, you can call the GetErrorInfo method for more information about the failure.

Remarks

Feedback events contain information such as messages generated from the sync, data for updating a progress bar, and synchronization cycle notifications.

As the dbmlsync server runs a synchronization it generates a series of events that contain information about the progress of the synchronization. These events are sent from the server to the DbmlsyncClient

class, which queues them. When the `GetEvent` method is called, the next event in the queue is returned if there is one waiting.

If there are no events waiting in the queue, this method waits until an event is available or until the specified timeout has expired before returning.

The types of events that are generated for a synchronization can be controlled using properties.

See also

- [“DBSC_GetEventRet enumeration” on page 249](#)
- [“DBSC_Event structure” on page 252](#)
- [“FreeEventInfo method” on page 237](#)
- [“SetProperty method” on page 241](#)
- [“GetErrorInfo method” on page 238](#)

GetProperty method

Retrieves the current value of a property.

Syntax

```
public virtual bool GetProperty(const char * name, char * value)
```

Parameters

- **name** The name of the property to retrieve. For a list of valid property names, see `SetProperty`.
- **value** A buffer of at least `DBSC_MAX_PROPERTY_LEN` bytes where the value of the property is stored.

Returns

True when the property was successfully received; otherwise, returns false. When false is returned, you can call the `GetErrorInfo` method for more information about the failure.

See also

- [“SetProperty method” on page 241](#)
- [“GetErrorInfo method” on page 238](#)

Init method

Initializes a `DbmsyncClient` class instance.

Syntax

```
public virtual bool Init(void )
```

Returns

True when the class instance is successfully initialized; otherwise, returns false. When false is returned, you can call the `GetErrorInfo` method for more information about the failure.

Remarks

This method must be called after instantiating the `DbmlSyncClient` class instance. Other `DbmlSyncClient` methods cannot be called until you have successfully initialized the instance.

See also

- [“InstantiateClient method” on page 240](#)
- [“GetErrorInfo method” on page 238](#)

InstantiateClient method

Creates an instance of the `dbmlsync` client class that can be used to control synchronizations.

Syntax

```
public static DbmlsyncClient * InstantiateClient(void )
```

Returns

A pointer to the new instance that has been created. Returns null when an error occurs.

Remarks

The pointer returned by this method can be used to call the remaining methods in the class. You can destroy the instance by calling the standard delete operator on the pointer.

Ping method

Sends a ping request to the `dbmlsync` server to check if the server is active and responding to requests.

Syntax

```
public virtual bool Ping(unsigned timeout)
```

Parameters

- **timeout** The maximum number of milliseconds to wait for the server to respond to the ping request. Use `DBSC_INFINITY` to wait indefinitely for a response.

Returns

True when a response to the ping request was received from the server; otherwise, returns false. When false is returned, you can call the `GetErrorInfo` method for more information about the failure.

Remarks

You must be connected to the server before calling this method.

See also

- [“GetErrorInfo method” on page 238](#)

SetProperty method

Sets various properties to modify the behavior of the class instance.

Syntax

```
public virtual bool SetProperty(const char * name, const char * value)
```

Parameters

- **name** The name of the property to set. For a list of valid property names, see table.
- **value** The value to set for the property. The string specified must contain less than DBCS_MAX_PROPERTY_LEN bytes.

Returns

True when the property was successfully set; otherwise, returns false. When false is returned, you can call the GetErrorInfo method for more information about the failure.

Remarks

Changes to property values only affect synchronization requests made after the property value was changed.

The **server path** property can be set to specify the directory from which the client should start dbmsync.exe when the StartServer method is called. When this property is not set, dbmsync.exe is found using the PATH environment variable. If there are multiple versions of SQL Anywhere installed on your computer, it is recommended that you specify the location of dbmsync.exe using the **server path** property because the PATH environment variable may locate a dbmsync executable from another installed version of SQL Anywhere. For example,

```
ret = cli->SetProperty("server path", "c:\\sa12\\bin32");
```

The properties control the types of events that are returned by the GetEvent method. By disabling events that you do not require you may be able to improve performance. An event type is enabled by setting the corresponding property to "1" and disabled by setting the property to "0".

The following is a table of available property names and the event types that each name controls:

Property name	Event types controlled	Default value
enable errors	DBSC_EVENTTYPE_ERROR_MSG	1
enable warnings	DBSC_EVENTTYPE_WARNING_MSG	1
enable info msgs	DBSC_EVENTTYPE_INFO_MSG	1

Property name	Event types controlled	Default value
enable progress	DBSC_EVENTTYPE_PROGRESS_INDEX	0
enable progress text	DBSC_EVENTTYPE_PROGRESS_TEXT	0
enable title	DBSC_EVENTTYPE_TITLE	0
enable sync start and done	DBSC_EVENTTYPE_SYNC_START DBSC_EVENTTYPE_SYNC_DONE	1
enable status	DBSC_EVENTTYPE_ML_CONNECT DBSC_EVENTTYPE_UPLOAD_COMMITTED DBSC_EVENTTYPE_DOWNLOAD_COMMITTED	1

See also

- [“StartServer method” on page 243](#)
- [“GetEvent method” on page 238](#)
- [“GetProperty method” on page 239](#)
- [“GetErrorInfo method” on page 238](#)

ShutdownServer method

Shuts down the dbmsync server to which the client is connected.

Syntax

```
public virtual bool ShutdownServer(DBSC_ShutdownType how)
```

Parameters

- **how** Indicates the urgency of the server shutdown. Supported values are listed in the DBSC_ShutdownType enumeration.

Returns

True when a shutdown request was successfully sent to the server; otherwise, returns false. When false is returned, you can call the GetErrorInfo method for more information about the failure.

Remarks

The Shutdown method returns immediately but there may be some delay before the server actually shuts down.

The WaitForServerShutdown method can be used to wait until the server actually shuts down.

Note

You should still use the Disconnect method after calling ShutdownServer.

See also

- [“DBSC_ShutdownType enumeration” on page 250](#)
- [“Disconnect method” on page 236](#)
- [“WaitForServerShutdown method” on page 245](#)
- [“GetErrorInfo method” on page 238](#)

StartServer method

Starts a new dbmsync server if one is not already listening on the specified port.

Syntax

```
public virtual bool StartServer(
    unsigned port,
    const char * cmdline,
    unsigned timeout,
    DBSC_StartType * starttype
)
```

Parameters

- **port** The TCP port to check for an existing dbmsync server. If a new server is started, it is set to listen on this port.
- **cmdline** A valid command line for starting a dbmsync server. The command line may contain only the following options which have the same meaning that they do for the dbmsync utility: -a, -c, -dl, -do, -ek, -ep, -k, -l, -o, -os, -ot, -p, -pc+, -pc-, -pd, -pp, -q, -qi, -qc, -sc, -sp, -uc, -ud, -ui, -um, -un, -ux, -v[*cnoprst*], -wc, -wh. The -c option must be specified.
- **timeout** The maximum time in milliseconds to wait after a dbmsync server is started for it to be ready to accept requests. Use DBSC_INFINITY to wait indefinitely for a response.
- **starttype** An out parameter set to indicate if the server has been located or started. If starttype is non-null on entry and StartServer returns true, then, on exit, the variable pointed to by starttype is set to a value from the DBSC_StartType enumeration.

Returns

True when the server was already running or successfully started; otherwise, returns false. When false is returned, you can call the GetErrorInfo method for more information about the failure.

Remarks

If a server is present, this method sets the starttype parameter to DBSC_SS_ALREADY_RUNNING and returns without further action. If no server is found, the method starts a new server using the options specified by the cmdline argument and waits for it to start accepting requests before returning.

On Windows Mobile devices, it is usually necessary to set the **server path** property before StartServer can be successfully called. The **server path** property does not need to be set in the following instances:

- Your application is in the same directory as *dbmsync.exe*.
- *dbmsync.exe* is in the Windows directory.

See also

- [“GetErrorInfo method” on page 238](#)

Sync method

Requests that the dbmsync server perform a synchronization.

Syntax

```
public virtual DBSC_SyncHdl Sync(  
    const char * profile_name,  
    const char * extra_opts  
)
```

Parameters

- **profile_name** The name of a synchronization profile defined in the remote database that contains the options for the synchronization. If profile_name is null then no profile is used and the extra_opts parameter should contain all the options for the synchronization.
- **extra_opts** A string formed according to the same rules used to define an option string for a synchronization profile, which is a string specified as a semicolon delimited list of elements of the form <option name>=<option value>=<option value>=<option value>. If profile_name is non-null then the options specified by extra_opts are added to those already in the synchronization profile specified by profile_name. If an option in the string already exists in the profile, then the value from the string replaces the value already stored in the profile. If profile_name is null then extra_opts should specify all the options for the synchronization. See [“CREATE SYNCHRONIZATION PROFILE statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#).

Returns

A DBSC_SyncHdl value which uniquely identifies this synchronization request and is only valid until the client disconnects from the server. Returns NULL_SYNCHDL if an error prevents the synchronization request from being created. When NULL_SYNCHDL is returned, you can call the GetErrorInfo method for more information about the failure.

Remarks

You must be connected to the server before calling this method. At least one of profile_name and extra_opts must be non-null.

The return value identifies the synchronization request and can be used to cancel the request or to process events returned by the synchronization

See also

- [“GetErrorInfo method” on page 238](#)

WaitForServerShutdown method

Returns when the server has shutdown or when the timeout expires, whichever comes first.

Syntax

```
public virtual bool WaitForServerShutdown(unsigned timeout)
```

Parameters

- **timeout** Indicates the maximum time in milliseconds to wait for the server to shutdown. Use DBSC_INFINITY to wait indefinitely for a response.

Returns

True when the method returned due to the server shutdown; otherwise, returns false. When false is returned, you can call the GetErrorInfo method for more information about the failure.

Remarks

WaitForServerShutdown can only be called after the ShutdownServer method is called.

See also

- [“GetErrorInfo method” on page 238](#)

DBSC_CancelRet enumeration

Indicates the result of a synchronization cancellation attempt.

Syntax

```
public enum DBSC_CancelRet
```

Members

Member name	Description	Value
DBSC_CANCEL_OK_QUEUED	Canceled a synchronization that was in the wait queue.	1
DBSC_CANCEL_OK_ACTIVE	Canceled an active synchronization.	2
DBSC_CANCEL_FAILED	Failed to cancel the synchronization.	3

See also

- [“CancelSync method” on page 234](#)

DBSC_ErrorType enumeration

Indicates the reason for a method call failure.

Syntax

```
public enum DBSC_ErrorType
```

Members

Member name	Description	Value
DBSC_ERR_OK	No error occurred.	1
DBSC_ERR_NOT_INITIALIZED	The class has not been initialized by calling the Init method.	2
DBSC_ERR_ALREADY_INITIALIZED	The Init method was called on a class that was already initialized.	3
DBSC_ERR_NOT_CONNECTED	No connection to a dbmlsync server is in place.	4
DBSC_ERR_CANT_RESOLVE_HOST	Cannot resolve host information.	5
DBSC_ERR_CONNECT_FAILED	Connection to the dbmlsync server has failed.	6
DBSC_ERR_INITIALIZING_TCP_LAYER	Error initializing TCP layer.	7
DBSC_ERR_ALREADY_CONNECTED	Connect method failed because a connection was already in place.	8
DBSC_ERR_PROTOCOL_ERROR	This is an internal error.	9
DBSC_ERR_CONNECTION_REJECTED	The connection was rejected by the dbmlsync server. str1 points to a string returned by the server which may provide more information about why the connection attempt was rejected.	10
DBSC_ERR_TIMED_OUT	The timeout expired while waiting for a response from the server.	11
DBSC_ERR_STILL_CONNECTED	Could not Fini the class because it is still connected to the server.	12
DBSC_ERR_SYNC_NOT_CANCELED	The server could not cancel the synchronization request, likely because the synchronization was already in progress.	14

Member name	Description	Value
DBSC_ERR_INVALID_VALUE	An invalid property value was passed to the Set-Property method.	15
DBSC_ERR_INVALID_PROP_NAME	The specified property name is not valid.	16
DBSC_ERR_VALUE_TOO_LONG	The property value is too long; properties must be less than DBCS_MAX_PROPERTY_LEN bytes long.	17
DBSC_ERR_SERVER_SIDE_ERROR	A server-side error occurred while cancelling or adding a sync. str1 points to a string returned by the server which may provide more information about the error.	18
DBSC_ERR_CREATE_PROCESS_FAILED	Unable to start a new dbmsync server.	20
DBSC_ERR_READ_FAILED	TCP error occurred while reading data from the dbmsync server.	21
DBSC_ERR_WRITE_FAILED	TCP error occurred while sending data to the dbmsync server.	22
DBSC_ERR_NO_SERVER_RESPONSE	Failed to receive a response from the server that is required to complete the requested action.	23
DBSC_ERR_UID_OR_PWD_TOO_LONG	The UID or PWD specified is too long.	24
DBSC_ERR_UID_OR_PWD_NOT_VALID	The UID or PWD specified is not valid.	25
DBSC_ERR_INVALID_PARAMETER	One of the parameters passed to the function was not valid.	26
DBSC_ERR_WAIT_FAILED	An error occurred while waiting for the server to shutdown.	27
DBSC_ERR_SHUTDOWN_NOT_CALLED	WaitForServerShutdown method was called without first calling the ShutdownServer method.	28

Member name	Description	Value
DBSC_ERR_NO_SYNC_ACK	A synchronization request was sent to the server but no acknowledgement was received; There is no way to know if the server received the request. hdl1 is the handle for the sync request that was sent. If the server received the request, this handle can be used to identify events for the synchronization retrieved using the GetEvent method.	29
DBSC_ERR_ACTIVE_SYNC_NOT_CANCELED	The server could not cancel the synchronization request because the synchronization was active.	30
DBSC_ERR_DEAD_SERVER	The dbmlsync server has encountered an error while starting up. The server is now shutting down. Use the dbmlsync -o option to log the error message to a file.	31

DBSC_EventType enumeration

Indicates the type of event generated by a synchronization.

Syntax

```
public enum DBSC_EventType
```

Members

Member name	Description	Value
DBSC_EVENTTYPE_ERROR_MSG	An error was generated by the synchronization; str1 points to the text of the error.	1
DBSC_EVENTTYPE_WARNING_MSG	A warning was generated by the synchronization; str1 points to the text of the warning.	2
DBSC_EVENTTYPE_INFO_MSG	An information message was generated by the synchronization; str1 points to the text of the message.	3
DBSC_EVENTTYPE_PROGRESS_INDEX	Provides information for updating a progress bar; val1 contains the new progress value. The percent done can be calculated by dividing val1 by 1000.	4

Member name	Description	Value
DBSC_EVENTTYPE_PROGRESS_TEXT	The text associated with the progress bar has been updated; the new value is pointed to by str1.	6
DBSC_EVENTTYPE_TITLE	The title for the synchronization window/control has changed; the new title is pointed to by str1.	7
DBSC_EVENTTYPE_SYNC_START	The synchronization has begun; there is no additional information associated with this event.	8
DBSC_EVENTTYPE_SYNC_DONE	The synchronization is complete; val1 contains the exit code from the synchronization. A 0 value indicates success. A non-zero value indicates that the synchronization failed.	9
DBSC_EVENTTYPE_ML_CONNECT	A connection to the MobiLink Server was established; str1 indicates the communication protocol being used and str2 contains the network protocol options used.	10
DBSC_EVENTTYPE_UPLOAD_COMMITTED	The MobiLink server confirmed that it successfully committed the upload to the consolidated database.	11
DBSC_EVENTTYPE_DOWNLOAD_COMMITTED	The download has been successfully committed in the remote database.	12

See also

- [“DBSC_Event structure” on page 252](#)
- [“GetEvent method” on page 238](#)

DBSC_GetEventRet enumeration

Indicates the result of an attempt to retrieve an event.

Syntax

```
public enum DBSC_GetEventRet
```

Members

Member name	Description	Value
DBSC_GETEVENT_OK	Indicates that an event was successfully retrieved.	1

Member name	Description	Value
DBSC_GETEVENT_TIMED_OUT	Indicates that the timeout expired without any event being available to return.	2
DBSC_GETEVENT_FAILED	Indicates that no event was returned because of an error condition.	3

See also

- [“GetEvent method” on page 238](#)

DBSC_ShutdownType enumeration

Indicates how urgently the server should be shut down.

Syntax

```
public enum DBSC_ShutdownType
```

Members

Member name	Description	Value
DBSC_SHUTDOWN_ON_EMPTY_QUEUE	Indicates that the server should complete any outstanding synchronization requests and then shutdown. Once the server receives the shutdown request, it does not accept any more synchronization requests.	1
DBSC_SHUTDOWN_CLEANLY	Indicates that the server should shutdown cleanly, as quickly as possible. If there are outstanding synchronization requests, they are not performed and if there is a running synchronization it may be interrupted.	2

See also

- [“ShutdownServer method” on page 242](#)

DBSC_StartType enumeration

Indicates the action taken during a dbmlsync server startup attempt.

Syntax

```
public enum DBSC_StartType
```

Members

Member name	Description	Value
DBSC_SS_STARTED	Indicates that a new dbmsync server was started.	1
DBSC_SS_AL-READY_RUNNING	Indicates that an existing dbmsync server was found, so no new server was started.	2

See also

- [“StartServer method” on page 243](#)

DBSC_ErrorInfo structure

Contains information about the failure of a previous method call.

Syntax

```
public struct DBSC_ErrorInfo
```

Members

Member name	Type	Description
hdl1	DBSC_SyncHdl	Contains additional information about the failure. The meaning of this information depends on the value of the type variable.
str1	const char *	Contains additional information about the failure. The meaning of this information depends on the value of the type variable.
str2	const char *	Contains additional information about the failure. The meaning of this information depends on the value of the type variable.
type	DBSC_ErrorType	Contains a value that indicates the reason for failure. Supported values are listed in the DBSC_ErrorType enumeration.
val1	long int	Contains additional information about the failure. The meaning of this information depends on the value of the type variable.
val2	long int	Contains additional information about the failure. The meaning of this information depends on the value of the type variable.

Remarks

str1, str2, val1, val2 and hdl1 contain additional information about the failure, and their meanings depend on the error type. The following error types use fields in this structure to store additional information:

- DBSC_ERR_CONNECTION_REJECTED
- DBSC_ERR_SERVER_SIDE_ERROR
- DBSC_ERR_NO_SYNC_ACK

See also

- [“DBSC_ErrorType enumeration” on page 246](#)

DBSC_Event structure

Contains information about an event generated by a synchronization.

Syntax

```
public struct DBSC_Event
```

Members

Member name	Type	Description
data	void *	Contains additional information about the failure. The meaning of this information depends on the value of the type variable.
hdl	DBSC_SyncHdl	Indicates the synchronization that generated the event. This value matches the value returned by the Sync method.
str1	const char *	Contains additional information about the failure. The meaning of this information depends on the value of the type variable.
str2	const char *	Contains additional information about the failure. The meaning of this information depends on the value of the type variable.
type	DBSC_EventType	Indicates the type of event being reported.
val1	long int	Contains additional information about the failure. The meaning of this information depends on the value of the type variable.
val2	long int	Contains additional information about the failure. The meaning of this information depends on the value of the type variable.

See also

- [“DBSC_EventType enumeration” on page 248](#)

Dbmsync .NET API reference

Namespace

iAnywhere.MobiLink.Client

Example

The sample below shows a typical application using the .NET version of the Dbmsync API to perform a synchronization and receive output events. The sample omits error handling for clarity. It is always good practice to check the return value from each API call.

```
using System;
using System.Collections.Generic;
using System.Text;
using iAnywhere.MobiLink.Client;

namespace ConsoleApplication6
{
    class Program
    {
        static void Main(string[] args)
        {
            DbmsyncClient cli1;
            DBSC_StartType st1;
            DBSC_Event ev1;
            UInt32 syncHdl;

            cli1 = DbmsyncClient.InstantiateClient();
            cli1.Init();

            // Setting the "server path" is usually required on Windows
            // Mobile/CE. In other environments the server path is usually
            // not required unless you SA install is not in your path or
            // you have multiple versions of the product installed
            cli1.SetProperty("server path", "d:\\sybase\\asa12\\bin32");

            cli1.StartServer(3426,
                "-c server=cons;dbn=reml;uid=dba;pwd=sql -ve+ -ot c:\\
                \\dbsync1.txt",
                5000, out st1);
            cli1.Connect(null, 3426, "dba", "sql");
            syncHdl = cli1.Sync("sp1", "");
            while (cli1.GetEvent(out ev1, 5000)
                == DBSC_GetEventRet.DBSC_GETEVENT_OK)
            {
                if (ev1.hdl == syncHdl)
                {
                    Console.WriteLine("Event Type : {0}", ev1.type);
                    if (ev1.type == DBSC_EventType.DBSC_EVENTTYPE_INFO_MSG)
                    {
                        Console.WriteLine("Info : {0}", ev1.str1);
                    }
                    if (ev1.type == DBSC_EventType.DBSC_EVENTTYPE_SYNC_DONE)
                    {
                        break;
                    }
                }
            }

            cli1.ShutdownServer(DBSC_ShutdownType.DBSC_SHUTDOWN_ON_EMPTY_QUEUE);
        }
    }
}
```

```

        cli1.WaitForServerShutdown(10000);
        cli1.Disconnect();
        cli1.Fini();
        Console.ReadLine();
    }
}

```

DbmlsyncClient class

Communicates using TCP/IP with a separate process, dbmlsync server, which performs a synchronization by connecting to the MobiLink server and the remote database.

Visual Basic syntax

```
Public Class DbmlsyncClient
```

C# syntax

```
public class DbmlsyncClient
```

Members

All members of DbmlsyncClient class, including all inherited members.

Name	Description
“CancelSync method”	Cancels a synchronization request.
“Connect method”	Opens a connection to a Dbmlsync server that is already running on this computer.
“Disconnect method”	Breaks the Dbmlsync server connection that was established with the Connect method.
“Fini method”	Frees all resources used by this class instance.
“GetErrorInfo method”	Retrieves additional information about the failure after a DbmlsyncClient class method returns a failed return code.
“GetEvent method”	Retrieves the next feedback event for synchronizations requested by the client.
“GetProperty method”	Retrieves the current value of a property.
“Init method”	Initializes a DbmlsyncClient class instance.
“InstantiateClient method”	Creates an instance of the dbmlsync client class that can be used to control synchronizations.

Name	Description
“Ping method”	Sends a ping request to the dbmlsync server to check if the server is active and responding to requests.
“SetProperty method”	Sets various properties to modify the behavior of the class instance.
“ShutdownServer method”	Shuts down the dbmlsync server to which the client is connected.
“StartServer method”	Starts a new dbmlsync server if one is not already listening on the specified port.
“Sync method”	Requests that the dbmlsync server perform a synchronization.
“WaitForServerShutdown method”	Returns when the server has shutdown or when the timeout expires, whichever comes first.

Remarks

Multiple clients can share the same dbmlsync server. However, each dbmlsync server can only synchronize a single remote database. Each remote database can have only one dbmlsync server synchronizing it.

The dbmlsync server performs one synchronization at a time. If the server receives a synchronization request while performing a synchronization, it queues that request and satisfies it later.

Status information generated by synchronizations is communicated back to the client application through the `GetEvent` method.

See also

- [“GetEvent method” on page 259](#)

CancelSync method

Cancels a synchronization request.

Overload list

Name	Description
“CancelSync(UInt32) method (deprecated)”	Allows a client to cancel a synchronization request previously made using the Sync method.
“CancelSync(UInt32, Boolean) method”	Allows a client to cancel a synchronization request previously made using the Sync method.

CancelSync(UInt32) method (deprecated)

Allows a client to cancel a synchronization request previously made using the Sync method.

Visual Basic syntax

```
Public Function CancelSync(ByVal hdl As UInt32) As Boolean
```

C# syntax

```
public Boolean CancelSync(UInt32 hdl)
```

Parameters

- **hdl** The synchronization handle returned by the Sync method when the synchronization was requested.

Returns

True when the synchronization request was successfully canceled; otherwise, returns false. When false is returned, you can call the GetErrorInfo method for more information about the failure.

Remarks

Only synchronization requests waiting to be serviced can be canceled. To stop a synchronization that has already begun, use the CancelSync(UInt32, Boolean) method.

You can use the ShutdownServer method and pass the DBSC_SHUTDOWN_CLEANLY type to cancel an active synchronization. Dbmlsync attempts to cancel the synchronization before shutting down. This task is equivalent of cancelling a synchronization using the DBTools interface.

A connection must be established to the server before this method can be used. This method cannot be used if the client has disconnected from the server since the Sync method was called.

See also

- [“GetErrorInfo method” on page 259](#)
- [“CancelSync method” on page 255](#)
- [“ShutdownServer method” on page 264](#)

CancelSync(UInt32, Boolean) method

Allows a client to cancel a synchronization request previously made using the Sync method.

Visual Basic syntax

```
Public Function CancelSync(  
    ByVal hdl As UInt32,  
    ByVal cancel_active As Boolean  
) As DBSC_CancelRet
```

C# syntax

```
public DBSC_CancelRet CancelSync(UInt32 hdl, Boolean cancel_active)
```

Parameters

- **hdl** The synchronization handle returned by the Sync method when the synchronization was requested.
- **cancel_active** When set to true, the request is cancelled if the synchronization has already begun. When set to false, the request is only cancelled if synchronization has not yet begun

Returns

A value from the DBSC_CancelRet enumeration. When DBSC_CANCEL_FAILED is returned, you can call the GetErrorInfo method for more information about the failure.

Remarks

Only synchronization requests waiting to be serviced can be canceled.

You can use the ShutdownServer method and pass the DBSC_SHUTDOWN_CLEANLY type to cancel an active synchronization. Dbmsync attempts to cancel the synchronization before shutting down. This task is equivalent of cancelling a synchronization using the DBTools interface.

A connection must be established to the server before this method can be used. This method cannot be used if the client has disconnected from the server since the Sync method was called.

See also

- [“DBSC_CancelRet enumeration” on page 268](#)
- [“GetErrorInfo method” on page 259](#)
- [“ShutdownServer method” on page 264](#)

Connect method

Opens a connection to a Dbmsync server that is already running on this computer.

Visual Basic syntax

```
Public Function Connect(  
    ByVal host As String,  
    ByVal port As Int32,  
    ByVal uid As String,  
    ByVal pwd As String  
) As Boolean
```

C# syntax

```
public Boolean Connect(String host, Int32 port, String uid, String pwd)
```

Parameters

- **host** This value is reserved. Specify null when using C#. Do not specify anything when using Visual Basic.
- **port** The TCP port on which the dbmsync server is listening. Use the same port value that you specified with the StartServer method.

- **uid** A valid database user id with DBA or REMOTE DBA authority on the remote database that is to be synchronized.
- **pwd** The database password for the user specified by uid.

Returns

True when a connection to the server was established; otherwise, returns false. When false is returned, you can call the `GetErrorInfo` method for more information about the failure.

Remarks

The database user id and password are used to validate whether this client has enough permissions to synchronize the database. When synchronizations are performed, the user id that was specified with the `-c` option when the `dbmlsync` server started is used.

See also

- [“StartServer method” on page 265](#)
- [“GetErrorInfo method” on page 259](#)

Disconnect method

Breaks the `Dbmlsync` server connection that was established with the `Connect` method.

Visual Basic syntax

```
Public Function Disconnect() As Boolean
```

C# syntax

```
public Boolean Disconnect()
```

Returns

True when the connection to the server has been broken; otherwise, returns false. When false is returned, you can call the `GetErrorInfo` method for more information about the failure.

Remarks

You should always call `Disconnect` when you are finished with a connection.

See also

- [“Connect method” on page 257](#)
- [“GetErrorInfo method” on page 259](#)

Fini method

Frees all resources used by this class instance.

Visual Basic syntax

```
Public Function Fini() As Boolean
```

C# syntax

```
public Boolean Fini()
```

Returns

True when the class instance is successfully finalized; otherwise, returns false. When false is returned, you can call the `GetErrorInfo` method for more information about the failure.

Remarks

This method must be called before you can delete the `DbmlSyncClient` class instance.

Note

You should use the `Disconnect` method to disconnect from any connected servers before finalizing the class instance.

See also

- [“Disconnect method” on page 258](#)
- [“GetErrorInfo method” on page 259](#)

GetErrorInfo method

Retrieves additional information about the failure after a `DbmlsyncClient` class method returns a failed return code.

Visual Basic syntax

```
Public Function GetErrorInfo() As DBSC_ErrorInfo
```

C# syntax

```
public DBSC_ErrorInfo GetErrorInfo()
```

Returns

A pointer to a `DBSC_ErrorInfo` structure that contains information about the failure. The contents of this structure may be overwritten the next time any class method is called.

See also

- [“DBSC_ErrorType enumeration” on page 268](#)
- [“DBSC_ErrorInfo structure” on page 274](#)
- [“GetErrorInfo method” on page 259](#)

GetEvent method

Retrieves the next feedback event for synchronizations requested by the client.

Visual Basic syntax

```
Public Function GetEvent(  
    ByVal ev As DBSC_Event,  
    ByVal timeout As UInt32  
) As DBSC_GetEventRet
```

C# syntax

```
public DBSC_GetEventRet GetEvent(out DBSC_Event ev, UInt32 timeout)
```

Parameters

- **ev** If the return value is DBSC_GETEVENT_OK then the *ev* parameter is filled with information about the event that has been retrieved.
- **timeout** Indicates the maximum time in milliseconds to wait if no event is immediately available to return. Use DbmlsyncClient.DBSC_INFINITY to wait indefinitely for a response.

Returns

A value from the DBSC_GetEventRet enumeration. When DBSC_GETEVENT_FAILED is returned, you can call the GetErrorInfo method for more information about the failure.

Remarks

Feedback events contain information such as messages generated from the sync, data for updating a progress bar, and synchronization cycle notifications.

As the dbmlsync server runs a synchronization it generates a series of events that contain information about the progress of the synchronization. These events are sent from the server to the DbmlsyncClient class, which queues them. When the GetEvent method is called, the next event in the queue is returned if there is one waiting.

If there are no events waiting in the queue, this method waits until an event is available or until the specified timeout has expired before returning.

The types of events that are generated for a synchronization can be controlled using properties.

See also

- [“DBSC_GetEventRet enumeration” on page 272](#)
- [“DBSC_Event structure” on page 275](#)
- [“SetProperty method” on page 262](#)
- [“GetErrorInfo method” on page 259](#)

GetProperty method

Retrieves the current value of a property.

Visual Basic syntax

```
Public Function GetProperty(  
    ByVal name As String,  
    ByVal value As String  
) As Boolean
```

C# syntax

```
public Boolean GetProperty(String name, out String value)
```

Parameters

- **name** The name of the property to retrieve. For a list of valid property names, see SetProperty.
- **value** On exit, the value of the property is stored in this variable.

Returns

True when the property was successfully received; otherwise, returns false. When false is returned, you can call the GetErrorInfo method for more information about the failure.

See also

- [“SetProperty method” on page 262](#)
- [“GetErrorInfo method” on page 259](#)

Init method

Initializes a DbmlsyncClient class instance.

Visual Basic syntax

```
Public Function Init() As Boolean
```

C# syntax

```
public Boolean Init()
```

Returns

True when the class instance is successfully initialized; otherwise, returns false. When false is returned, you can call the GetErrorInfo method for more information about the failure.

Remarks

This method must be called after instantiating the DbmlSyncClient class instance. Other DbmlSyncClient methods cannot be called until you have successfully initialized the instance.

See also

- [“InstantiateClient method” on page 262](#)
- [“GetErrorInfo method” on page 259](#)

InstantiateClient method

Creates an instance of the dbmsync client class that can be used to control synchronizations.

Visual Basic syntax

```
Public Shared Function InstantiateClient() As DbmsyncClient
```

C# syntax

```
public static DbmsyncClient InstantiateClient()
```

Returns

The created DbmsyncClient instance. Returns null when an error occurs.

Remarks

The object returned by this method can be used to call the remaining methods in the class.

Ping method

Sends a ping request to the dbmsync server to check if the server is active and responding to requests.

Visual Basic syntax

```
Public Function Ping(ByVal timeout As UInt32) As Boolean
```

C# syntax

```
public Boolean Ping(UInt32 timeout)
```

Parameters

- **timeout** The maximum number of milliseconds to wait for the server to respond to the ping request. Use DbmsyncClient.DBSC_INFINITY to wait indefinitely for a response.

Returns

True when a response to the ping request was received from the server; otherwise, returns false. When false is returned, you can call the GetErrorInfo method for more information about the failure.

Remarks

You must be connected to the server before calling this method.

See also

- [“GetErrorInfo method” on page 259](#)

SetProperty method

Sets various properties to modify the behavior of the class instance.

Visual Basic syntax

```
Public Function SetProperty(
    ByVal name As String,
    ByVal value As String
) As Boolean
```

C# syntax

```
public Boolean SetProperty(String name, String value)
```

Parameters

- **name** The name of the property to set. For a list of valid property names, see table.
- **value** The value to set for the property.

Returns

True when the property was successfully set; otherwise, returns false. When false is returned, you can call the GetErrorInfo method for more information about the failure.

Remarks

Changes to property values only affect synchronization requests made after the property value was changed.

The **server path** property can be set to specify the directory from which the client should start dbmsync.exe when the StartServer method is called. When this property is not set, dbmsync.exe is found using the PATH environment variable. If there are multiple versions of SQL Anywhere installed on your computer, it is recommended that you specify the location of dbmsync.exe using the **server path** property because the PATH environment variable may locate a dbmsync executable from another installed version of SQL Anywhere. For example,

```
ret = cli->SetProperty("server path", "c:\\sa12\\bin32");
```

The properties control the types of events that are returned by the GetEvent method. By disabling events that you do not require you may be able to improve performance. An event type is enabled by setting the corresponding property to "1" and disabled by setting the property to "0".

The following is a table of available property names and the event types that each name controls:

Property name	Event types controlled	Default value
enable errors	DBSC_EVENTTYPE_ERROR_MSG	1
enable warnings	DBSC_EVENTTYPE_WARNING_MSG	1
enable info msgs	DBSC_EVENTTYPE_INFO_MSG	1
enable progress	DBSC_EVENTTYPE_PROGRESS_INDEX	0
enable progress text	DBSC_EVENTTYPE_PROGRESS_TEXT	0
enable title	DBSC_EVENTTYPE_TITLE	0

Property name	Event types controlled	Default value
enable sync start	DBSC_EVENTTYPE_SYNC_START	1
enable sync done	DBSC_EVENTTYPE_SYNC_DONE	1
enable sync start and done	DBSC_EVENTTYPE_SYNC_START DBSC_EVENTTYPE_SYNC_DONE	1
enable status	DBSC_EVENTTYPE_ML_CONNECT DBSC_EVENTTYPE_UPLOAD_COMMITTED DBSC_EVENTTYPE_DOWNLOAD_COMMITTED	1

See also

- [“StartServer method” on page 265](#)
- [“GetEvent method” on page 259](#)
- [“GetProperty method” on page 260](#)
- [“GetErrorInfo method” on page 259](#)

ShutdownServer method

Shuts down the dbmsync server to which the client is connected.

Visual Basic syntax

```
Public Function ShutdownServer(  
    ByVal how As DBSC_ShutdownType  
) As Boolean
```

C# syntax

```
public Boolean ShutdownServer(DBSC_ShutdownType how)
```

Parameters

- **how** Indicates the urgency of the server shutdown. Supported values are listed in the DBSC_ShutdownType enumeration.

Returns

True when a shutdown request was successfully sent to the server; otherwise, returns false. When false is returned, you can call the GetErrorInfo method for more information about the failure.

Remarks

The Shutdown method returns immediately but there may be some delay before the server actually shuts down.

The `WaitForServerShutdown` method can be used to wait until the server actually shuts down.

Note

You should still use the `Disconnect` method after calling `ShutdownServer`.

See also

- [“DBSC_ShutdownType enumeration” on page 273](#)
- [“Disconnect method” on page 258](#)
- [“WaitForServerShutdown method” on page 267](#)
- [“GetErrorInfo method” on page 259](#)

StartServer method

Starts a new dbmsync server if one is not already listening on the specified port.

Visual Basic syntax

```
Public Function StartServer(
    ByVal port As Int32,
    ByVal cmdline As String,
    ByVal timeout As UInt32,
    ByVal starttype As DBSC_StartType
) As Boolean
```

C# syntax

```
public Boolean StartServer(
    Int32 port,
    String cmdline,
    UInt32 timeout,
    out DBSC_StartType starttype
)
```

Parameters

- **port** The TCP port to check for an existing dbmsync server. If a new server is started, it is set to listen on this port.
- **cmdline** A valid command line for starting a dbmsync server. The command line may contain only the following options which have the same meaning that they do for the dbmsync utility: -a, -c, -dl, -do, -ek, -ep, -k, -l, -o, -os, -ot, -p, -pc+, -pc-, -pd, -pp, -q, -qi, -qc, -sc, -sp, -uc, -ud, -ui, -um, -un, -ux, -v[*cnoprst*], -wc, -wh. The -c option must be specified.
- **timeout** The maximum time in milliseconds to wait after a dbmsync server is started for it to be ready to accept requests. Use `DbmsyncClient.DBSC_INFINITY` to wait indefinitely for a response.
- **starttype** An out parameter set to indicate if the server has been located or started. If `starttype` is non-null on entry and `StartServer` returns true, then, on exit, the variable pointed to by `starttype` is set to a value from the `DBSC_StartType` enumeration.

Returns

True when the server was already running or successfully started; otherwise, returns false. When false is returned, you can call the `GetErrorInfo` method for more information about the failure.

Remarks

If a server is present, this method sets the `starttype` parameter to `DBSC_SS_ALREADY_RUNNING` and returns without further action. If no server is found, the method starts a new server using the options specified by the `cmdline` argument and waits for it to start accepting requests before returning.

On Windows Mobile devices, it is usually necessary to set the **server path** property before `StartServer` can be successfully called. The **server path** property does not need to be set in the following instances:

- Your application is in the same directory as `dbmsync.exe`.
- `dbmsync.exe` is in the Windows directory.

See also

- [“GetErrorInfo method” on page 259](#)

Sync method

Requests that the `dbmsync` server perform a synchronization.

Visual Basic syntax

```
Public Function Sync(  
    ByVal syncName As String,  
    ByVal opts As String  
) As UInt32
```

C# syntax

```
public UInt32 Sync(String syncName, String opts)
```

Parameters

- **syncName** The name of a synchronization profile defined in the remote database that contains the options for the synchronization. If `syncName` is null then no profile is used and the `opts` parameter should contain all the options for the synchronization.
- **opts** A string formed according to the same rules used to define an option string for a synchronization profile, which is a string specified as a semicolon delimited list of elements of the form `<option name>=<option value>`. If `syncName` is non-null then the options specified by `opts` are added to those already in the synchronization profile specified by `syncName`. If an option in the string already exists in the profile, then the value from the string replaces the value already stored in the profile. If the `syncName` is null then `opts` should specify all the options for the synchronization. See [“CREATE SYNCHRONIZATION PROFILE statement \[MobiLink\]” \[SQL Anywhere Server - SQL Reference\]](#).

Returns

An integer value which uniquely identifies this synchronization request and is only valid until the client disconnects from the server. Returns `NULL_SYNCHDL` if an error prevents the synchronization request from being created. When `NULL_SYNCHDL` is returned, you can call the `GetErrorInfo` method for more information about the failure.

Remarks

You must be connected to the server before calling this method. At least one of `syncName` and `opts` must be non-null.

The return value identifies the synchronization request and can be used to cancel the request or to process events returned by the synchronization.

The following C# example demonstrates how to display error codes after invoking the `Sync` method.

```
// Insert code to initialize the synchronization client.

UInt32 request = syncClient.Sync("syncName", null);
if (request == DbmsyncClient.NULL_SYNCHDL) {
    string error_code = syncClient.GetErrorInfo().type.ToString();
    MessageBox.Show(error_code, "Sync Error");
}
```

See also

- [“GetErrorInfo method” on page 259](#)

WaitForServerShutdown method

Returns when the server has shutdown or when the timeout expires, whichever comes first.

Visual Basic syntax

```
Public Function WaitForServerShutdown(  
    ByVal timeout As UInt32  
) As Boolean
```

C# syntax

```
public Boolean WaitForServerShutdown(UInt32 timeout)
```

Parameters

- **timeout** Indicates the maximum time in milliseconds to wait for the server to shutdown. Use `DbmsyncClient.DBSC_INFINITY` to wait indefinitely for a response.

Returns

True when the method returned due to the server shutdown; otherwise, returns false. When false is returned, you can call the `GetErrorInfo` method for more information about the failure.

Remarks

`WaitForServerShutdown` can only be called after the `ShutdownServer` method is called.

See also

- [“GetErrorInfo method” on page 259](#)

DBSC_CancelRet enumeration

Indicates the result of a synchronization cancellation attempt.

Visual Basic syntax

```
Public Enum DBSC_CancelRet
```

C# syntax

```
public enum DBSC_CancelRet
```

Members

Member name	Description	Value
DBSC_CANCEL_OK_QUEUED	Canceled a synchronization that was in the wait queue.	1
DBSC_CANCEL_OK_ACTIVE	Canceled an active synchronization.	2
DBSC_CANCEL_FAILED	Failed to cancel the synchronization.	3

See also

- [“CancelSync method” on page 255](#)

DBSC_ErrorType enumeration

Indicates the reason for a method call failure.

Visual Basic syntax

```
Public Enum DBSC_ErrorType
```

C# syntax

```
public enum DBSC_ErrorType
```

Members

Member name	Description	Value
DBSC_ERR_OK	No error occurred.	1

Member name	Description	Value
DBSC_ERR_NOT_INITIALIZED	The class has not been initialized by calling the Init method.	2
DBSC_ERR_ALREADY_INITIALIZED	The Init method was called on a class that was already initialized.	3
DBSC_ERR_NOT_CONNECTED	No connection to a dbmsync server is in place.	4
DBSC_ERR_CANT_RESOLVE_HOST	Cannot resolve host information.	5
DBSC_ERR_CONNECT_FAILED	Connection to the dbmsync server has failed.	6
DBSC_ERR_INITIALIZING_TCP_LAYER	Error initializing TCP layer.	7
DBSC_ERR_ALREADY_CONNECTED	Connect method failed because a connection was already in place.	8
DBSC_ERR_PROTOCOL_ERROR	This is an internal error.	9
DBSC_ERR_CONNECTION_REJECTED	The connection was rejected by the dbmsync server. str1 points to a string returned by the server which may provide more information about why the connection attempt was rejected.	10
DBSC_ERR_TIMED_OUT	The timeout expired while waiting for a response from the server.	11
DBSC_ERR_STILL_CONNECTED	Could not Fini the class because it is still connected to the server.	12
DBSC_ERR_SYNC_NOT_CANCELED	The server could not cancel the synchronization request, likely because the synchronization was already in progress.	14
DBSC_ERR_INVALID_VALUE	An invalid property value was passed to the SetProperty method.	15
DBSC_ERR_INVALID_PROP_NAME	The specified property name is not valid.	16
DBSC_ERR_VALUE_TOO_LONG	The property value is too long; properties must be less than DBCS_MAX_PROPERTY_LEN bytes long.	17

Member name	Description	Value
DBSC_ERR_SERVER_SIDE_ERROR	A server-side error occurred while cancelling or adding a sync. str1 points to a string returned by the server which may provide more information about the error.	18
DBSC_ERR_CREATE_PROCESS_FAILED	Unable to start a new dbmsync server.	20
DBSC_ERR_READ_FAILED	TCP error occurred while reading data from the dbmsync server.	21
DBSC_ERR_WRITE_FAILED	TCP error occurred while sending data to the dbmsync server.	22
DBSC_ERR_NO_SERVER_RESPONSE	Failed to receive a response from the server that is required to complete the requested action.	23
DBSC_ERR_UID_OR_PWD_TOO_LONG	The UID or PWD specified is too long.	24
DBSC_ERR_UID_OR_PWD_NOT_VALID	The UID or PWD specified is not valid.	25
DBSC_ERR_INVALID_PARAMETER	One of the parameters passed to the function was not valid.	26
DBSC_ERR_WAIT_FAILED	An error occurred while waiting for the server to shutdown.	27
DBSC_ERR_SHUTDOWN_NOT_CALLED	WaitForServerShutdown method was called without first calling the ShutdownServer method.	28
DBSC_ERR_NO_SYNC_ACK	A synchronization request was sent to the server but no acknowledgement was received; There is no way to indicate that the server received the request. hdl1 is the handle for the sync request that was sent. If the server received the request, this handle can be used to identify events for the synchronization retrieved using the GetEvent method.	29

Member name	Description	Value
DBSC_ERR_ACTIVE_SYNC_NOT_CANCELED	The server could not cancel the synchronization request because the synchronization was active.	30
DBSC_ERR_DEAD_SERVER	The dbmlsync server has encountered an error while starting up. The server is now shutting down. Use the dbmlsync -o option to log the error message to a file.	31

DBSC_EventType enumeration

Indicates the type of event generated by a synchronization.

Visual Basic syntax

```
Public Enum DBSC_EventType
```

C# syntax

```
public enum DBSC_EventType
```

Members

Member name	Description	Value
DBSC_EVENTTYPE_ERROR_MSG	An error was generated by the synchronization; str1 contains the text of the error.	1
DBSC_EVENTTYPE_WARNING_MSG	A warning was generated by the synchronization; str1 contains the text of the warning.	2
DBSC_EVENTTYPE_INFO_MSG	An information message was generated by the synchronization; str1 contains the text of the message.	3
DBSC_EVENTTYPE_PROGRESS_INDEX	Provides information for updating a progress bar; val1 contains the new progress value. The percent done can be calculated by dividing val1 by 1000.	4

Member name	Description	Value
DBSC_EVENTTYPE_PROGRESS_TEXT	The text associated with the progress bar has been updated and str1 contains the new value.	5
DBSC_EVENTTYPE_TITLE	The title for the synchronization window/control has changed and str1 contains the new title.	6
DBSC_EVENTTYPE_SYNC_START	The synchronization has begun; there is no additional information associated with this event.	7
DBSC_EVENTTYPE_SYNC_DONE	The synchronization is complete; val1 contains the exit code from the synchronization. A 0 value indicates success. A non-zero value indicates that the synchronization failed.	8
DBSC_EVENTTYPE_ML_CONNECT	A connection to the MobiLink Server was established; str1 indicates the communication protocol being used and str2 contains the network protocol options used.	10
DBSC_EVENTTYPE_UPLOAD_COMMITTED	The MobiLink server confirmed that it successfully committed the upload to the consolidated database.	11
DBSC_EVENTTYPE_DOWNLOAD_COMMITTED	The download has been successfully committed in the remote database.	12

See also

- [“DBSC_Event structure” on page 275](#)
- [“GetEvent method” on page 259](#)

DBSC_GetEventRet enumeration

Indicates the result of an attempt to retrieve an event.

Visual Basic syntax

```
Public Enum DBSC_GetEventRet
```

C# syntax

```
public enum DBSC_GetEventRet
```

Members

Member name	Description	Value
DBSC_GETEVENT_OK	Indicates that an event was successfully retrieved.	1
DBSC_GETE- VENT_TIMED_OUT	Indicates that the timeout expired without any event being available to return.	2
DBSC_GETE- VENT_FAILED	Indicates that no event was returned because of an error condition.	3

See also

- [“GetEvent method” on page 259](#)

DBSC_ShutdownType enumeration

Indicates how urgently the server should be shut down.

Visual Basic syntax

```
Public Enum DBSC_ShutdownType
```

C# syntax

```
public enum DBSC_ShutdownType
```

Members

Member name	Description	Value
DBSC_SHUT- DOWN_ON_EMP- TY_QUEUE	Indicates that the server should complete any outstanding synchronization requests and then shutdown. Once the server receives the shutdown request, it does not accept any more synchronization requests.	1
DBSC_SHUT- DOWN_CLEANLY	Indicates that the server should shutdown cleanly, as quickly as possible. If there are outstanding synchronization requests, they are not performed and if there is a running synchronization it may be interrupted.	2

See also

- [“ShutdownServer method” on page 264](#)

DBSC_StartType enumeration

Indicates the action taken during a dbmlsync server startup attempt.

Visual Basic syntax

```
Public Enum DBSC_StartType
```

C# syntax

```
public enum DBSC_StartType
```

Members

Member name	Description	Value
DBSC_SS_STARTED	Indicates that a new dbmlsync server was started.	1
DBSC_SS_ALREADY_RUNNING	Indicates that an existing dbmlsync server was found, so no new server was started.	2

See also

- [“StartServer method” on page 265](#)

DBSC_ErrorInfo structure

Contains information about the failure of a previous method call.

Visual Basic syntax

```
Structure DBSC_ErrorInfo
```

C# syntax

```
public struct DBSC_ErrorInfo
```

Members

Member name	Type	Description
hdl1	UInt32	Contains additional information about the failure. The meaning of this information depends on the value of the type variable.
str1	String	Contains additional information about the failure. The meaning of this information depends on the value of the type variable.

Member name	Type	Description
str2	String	Contains additional information about the failure. The meaning of this information depends on the value of the type variable.
type	DBSC_Error- Type	Contains a value that indicates the reason for failure. Supported values are listed in the DBSC_ErrorType enumeration.
val1	Int32	Contains additional information about the failure. The meaning of this information depends on the value of the type variable.
val2	Int32	Contains additional information about the failure. The meaning of this information depends on the value of the type variable.

Remarks

str1, str2, val1, val2 and hdl1 contain additional information about the failure, and their meanings depend on the error type. The following error types use fields in this structure to store additional information:

- DBSC_ERR_CONNECTION_REJECTED
- DBSC_ERR_SERVER_SIDE_ERROR
- DBSC_ERR_NO_SYNC_ACK

See also

- [“DBSC_ErrorType enumeration” on page 268](#)

DBSC_Event structure

Contains information about an event generated by a synchronization.

Visual Basic syntax

```
Structure DBSC_Event
```

C# syntax

```
public struct DBSC_Event
```

Members

Member name	Type	Description
hdl	UInt32	Indicates the synchronization that generated the event. This value matches the value returned by the Sync method.
str1	String	Contains additional information about the failure. The meaning of this information depends on the value of the type variable.
str2	String	Contains additional information about the failure. The meaning of this information depends on the value of the type variable.
type	DBSC_EventType	Indicates the type of event being reported.
val1	Int32	Contains additional information about the failure. The meaning of this information depends on the value of the type variable.
val2	Int32	Contains additional information about the failure. The meaning of this information depends on the value of the type variable.

See also

- [“DBSC_EventType enumeration” on page 271](#)

Dbmlsync integration component (removed)

Note

The Dbmlsync integration component has been removed in version 12. In its place, use the dbmlsync programming interface. See [“Dbmlsync API” on page 90](#).

Introduction to Dbmlsync integration component (removed)

Note

The Dbmlsync integration component has been removed in version 12. In its place, use the dbmlsync programming interface. See [“Dbmlsync API” on page 90](#).

The Dbmsync integration component is an ActiveX that you can use to add synchronization to your applications. It provides a set of properties, events, and methods to regulate the behavior of SQL Anywhere clients.

The Dbmsync integration component is available in two forms, both of which expose the same properties, events and methods:

- A visual component that provides an easy way to integrate the standard dbmsync user interface into your applications.
- A non-visual component that allows you to access the component's functionality with no user interface or with a custom user interface that you create yourself.

Using the Dbmsync integration component, your application can initiate synchronization, receive information about the progress of a synchronization, and implement special processing based on synchronization events.

DBTools interface for dbmsync

As an alternative to the Dbmsync integration component, you can use DBTools interface for dbmsync.

See “[Database tools interface \(DBTools\)](#)” [*SQL Anywhere Server - Programming*].

Supported platforms (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See “[Dbmsync API](#)” on page 90.

You can use the Dbmsync integration component on all MobiLink supported Windows operating systems, including Windows Mobile versions supporting ActiveX.

Supported development environments include Microsoft Visual Basic 6.0, eMbedded Visual Basic, and Visual Studio.

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

Setting up the Dbmsync integration component (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See “[Dbmsync API](#)” on page 90.

The Dbmsync integration component is an ActiveX and can be used in a wide variety of programming environments. You should consult the documentation for your programming environment for information about how to set it up.

Dbmsync integration component methods (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

The following are methods implemented by the DbmsyncCOM.Dbmsync class.

Run method (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

Begins one or more synchronizations using dbmsync command line options.

Syntax

Run(ByVal *cmdLine* As String)
Member of **DbmsyncCOM.Dbmsync**

Parameters

cmdLine A string specifying dbmsync options.

Remarks

For a list of options, see [“dbmsync syntax” on page 93](#).

The run method returns immediately and does not wait for the synchronization to complete. You can use the DoneExecution event to determine when your synchronization is complete.

The cmdLine parameter should contain the same options you would use if you were performing a synchronization with the dbmsync command line utility. For example, the following command line and Run method invocation are equivalent:

```
dbmsync -c uid=DBA;pwd=sql
```

```
dbmsync1.Run "-c uid=DBA;pwd=sql"
```

Example

The following example initiates a synchronization for a remote database called remotel.

```
dbmsync1.Run "-c server=remotel;uid=DBA;pwd=sql"
```

Stop method (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

Requests active synchronizations to terminate.

Syntax

Stop()
Member of **DbmsyncCOM.Dbmsync**

Remarks

The Stop method issues a request to terminate any active synchronizations. It returns immediately.

The stop button built into the visual Dbmsync integration component automatically invokes this method.

Example

The following example stops synchronizations being run by the Dbmsync integration component instance dbmsync1.

```
dbmsync1.Stop
```

Dbmsync integration component properties (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

Dbmsync integration component properties let you customize the behavior of the component and examine the state of a running synchronization.

Path property (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

Specifies the location of *dbmsync.exe*.

Syntax

Public Property **Path()** As String
Member of **DbmsyncCOM.Dbmsync**

Remarks

You do not need to set this property if *dbmsync.exe* is located in a directory specified by the Windows PATH environment variable.

Example

The following example sets the path of a Dbmsync integration component instance.

```
dbmsync1.Path = "C:\Program Files\SQL Anywhere 12\bin32"
```

UploadEventsEnabled property (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

Enables the UploadRow event.

Syntax

Public Property **UploadEventsEnabled()** As Boolean
Member of **DbmsyncCOM.Dbmsync**

Remarks

If you handle the UploadRow event, you should set this property to true. The default is false, which disables the UploadRow event. Setting the property to true reduces performance.

See [“UploadRow event \(removed\)” on page 299](#).

Example

The following example sets UploadEventsEnabled to true:

```
dbmsync1.UploadEventsEnabled = True
```

DownloadEventsEnabled property (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

Enables the DownloadRow event.

Syntax

Public Property **DownloadEventsEnabled()** As Boolean
Member of **DbmsyncCOM.Dbmsync**

Remarks

If you handle the DownloadRow event, you should set this property to true. The default is false, which disables the DownloadRow event. Setting the property to true reduces performance.

See [“DownloadRow event \(removed\)” on page 290](#).

Example

The following example sets DownloadEventsEnabled to true:

```
dbmsync1.DownloadEventsEnabled = True
```

ErrorMessageEnabled property (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

Prevents the Message event from being called for messages of type MsgError.

Syntax

Public Property **ErrorMessageEnabled**() As Boolean
Member of **DbmsyncCOM.Dbmsync**

Remarks

If you do not handle error information in the Message event, you should set this property to false to improve performance. The default is true, which enables messages of type MsgError to trigger the Message event.

See [“Message event \(removed\)” on page 294](#).

Example

The following example sets ErrorMessageEnabled to false:

```
dbmsync1.ErrorMessageEnabled = False
```

WarningMessageEnabled property (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

Prevents the Message event from being called for messages of type MsgWarning.

Syntax

Public Property **WarningMessageEnabled()** As Boolean
Member of **DbmlsyncCOM.Dbmlsync**

Remarks

If you do not handle warning information in the Message event, you should set this property to false to improve performance. The default is true, which enables messages of type `MsgWarning` to trigger the Message event.

See [“Message event \(removed\)” on page 294](#).

Example

The following example sets `WarningMessageEnabled` to false:

```
dbmlsync1.WarningMessageEnabled = False
```

InfoMessageEnabled property (removed)

Note

The `Dbmlsync` integration component has been removed in version 12. In its place, use the `dbmlsync` programming interface. See [“Dbmlsync API” on page 90](#).

Prevents the Message event from being called for messages of type `MsgInfo`.

Syntax

Public Property **InfoMessageEnabled()** As Boolean
Member of **DbmlsyncCOM.Dbmlsync**

Remarks

If you do not handle general progress information in the Message event, you should set this property to false to improve performance. The default is true, which enables messages of type `MsgInfo` to trigger the Message event.

See [“Message event \(removed\)” on page 294](#).

Example

The following example sets `InfoMessageEnabled` to false:

```
dbmlsync1.InfoMessageEnabled = False
```

DetailedInfoMessageEnabled property (removed)

Note

The `Dbmlsync` integration component has been removed in version 12. In its place, use the `dbmlsync` programming interface. See [“Dbmlsync API” on page 90](#).

Prevents the Message event from being called for messages of type MsgDetailedInfo.

Syntax

Public Property **DetailedInfoMessageEnabled**() As Boolean
Member of **DbmsyncCOM.Dbmsync**

Remarks

If you do not handle detailed progress information in the Message event, you should set this property to false to improve performance. The default is true, which enables messages of type MsgDetailedInfo to trigger the Message event.

See [“Message event \(removed\)” on page 294](#).

Example

The following example sets DetailedInfoMessageEnabled to false:

```
dbmsync1.DetailedInfoMessageEnabled = False
```

UseVB6Types property (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

If you are using Visual Basic 6, set this property to true to simplify handling of row data returned by the UploadRow and DownloadRow events.

Syntax

Public Property **DetailedInfoMessageEnabled**() As Boolean
Member of **DbmsyncCOM.Dbmsync**

Remarks

Visual Basic 6 does not support unsigned 32 bit values and any 64 bit values. Data of these types may be returned by the ColumnValue property of an IRowTransferData object. When UseVB6Types is set to true, data of these types is converted to other types supported by Visual Basic 6 for easier processing. UInt32 values are converted to double; 64 bit values are converted to strings.

See also

- [“IRowTransferData interface \(removed\)” on page 300](#)
- [“UploadRow event \(removed\)” on page 299](#)
- [“DownloadRow event \(removed\)” on page 290](#)

Example

The following example enables data type coercion for a Dbmsync integration component instance used in Visual Basic 6.0:

```
dbmsync1.UseVB6Types = True
```

ExitCode property (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

Returns the exit code from synchronizations started by the most recent Run method invocation.

Syntax

Public Property **ExitCode()** As Integer
Member of **DbmsyncCOM.Dbmsync**

Remarks

The ExitCode property returns the exit code for the synchronizations started by the last Run method invocation. 0 indicates successful synchronizations. Any other value indicates that a synchronization failed.

Note

Retrieving the value of this property before the DoneExecution event is triggered may result in a meaningless exit code value.

Example

The following example displays the exit code from the most recent synchronization attempt when the DoneExecution event is triggered.

```
Private Sub dbmsync1_DoneExecution() Handles dbmsync1.DoneExecution
    MsgBox(dbmsync1.ExitCode)
End Sub
```

EventChannelSize property (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

Specifies the size of an internal buffer used for processing method calls.

Syntax

Public Property **EventChannelSize()** As Integer
Member of **DbmsyncCOM.Dbmsync**

Remarks

Most users never have to change this property.

DispatchChannelSize property (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

Specifies the size of an internal buffer used for processing event information.

Syntax

Public Property **DispatchChannelSize()** As Integer
Member of **DbmsyncCOM.Dbmsync**

Remarks

Most users never have to change this property.

Dbmsync integration component events (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

Events provide a mechanism for client applications to receive and act on information about the progress of a synchronization.

BeginDownload event (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

The BeginDownload event is triggered at the beginning of the download stage of a synchronization.

Syntax

Public Event **BeginDownload()**
Member of **DbmsyncCOM.Dbmsync**

Remarks

Use this event to add custom actions at the beginning of the download stage of a synchronization.

Example

The following Visual Basic .NET example outputs a message when the BeginDownload event is triggered.

```
Private Sub dbmsync1_BeginDownload()  
Handles dbmsync1.BeginDownload  
  
    MsgBox("Beginning Download")  
  
End Sub
```

BeginLogScan event (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

The BeginLogScan event is triggered immediately before dbmsync scans the transaction log to assemble the upload. This event is not fired for scripted uploads.

Syntax

Public Event **BeginLogScan**(ByVal *rescanLog* As Boolean)
Member of **DbmsyncCOM.Dbmsync**

Parameters

rescanLog If this is the first time the transaction log has been scanned for this synchronization, the value is false; otherwise it is true. The log is scanned twice when the MobiLink server and dbmsync have different information about where scanning should begin.

Remarks

Use this event to add custom actions immediately before the transaction log is scanned for upload.

Example

The following Visual Basic .NET example outputs a message when the BeginLogScan event is triggered.

```
Private Sub dbmsync1_BeginLogScan(  
    ByVal rescanLog As Boolean  
)  
Handles dbmsync1.BeginLogScan  
  
    MsgBox("Begin Log Scan")  
  
End Sub
```

BeginSynchronization event (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

The BeginSynchronization event is triggered at the beginning of each synchronization.

Syntax

```
Public Event BeginSynchronization( _
    ByVal userName As String, _
    ByVal pubNames As String _
)
```

Member of **DbmlsyncCOM.Dbmlsync**

Parameters

userName The MobiLink user for which you are synchronizing.

pubNames The publication being synchronized. If there is more than one publication this is a comma-separated list.

Remarks

Use this event to add custom actions at the beginning of a synchronization.

Example

The following Visual Basic .NET example outputs a message when the BeginSynchronization event is triggered. The message outputs the user and publication names.

```
Private Sub dbmlsync1_BeginSynchronization(
    ByVal userName As String,
    ByVal pubNames As String
)
    Handles dbmlsync1.BeginSynchronization
        MsgBox("Beginning synchronization for: " + userName _
            + " publication: " + pubNames)
End Sub
```

BeginUpload event (removed)**Note**

The Dbmlsync integration component has been removed in version 12. In its place, use the dbmlsync programming interface. See [“Dbmlsync API” on page 90](#).

The BeginUpload event is triggered immediately before the transmission of the upload.

Syntax

```
Public Event BeginUpload( )
```

Member of **DbmlsyncCOM.Dbmlsync**

Remarks

Use this event to add custom actions immediately before the transmission of the upload to the MobiLink server.

Example

The following Visual Basic .NET example outputs a message when the BeginUpload event is triggered.

```
Private Sub dbmsync1_BeginUpload()  
Handles dbmsync1.BeginUpload  
  
    MsgBox("Begin Upload")  
  
End Sub
```

ConnectMobilink event (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

The ConnectMobilink event is triggered immediately before the component connects to the MobiLink server.

Syntax

Public Event **ConnectMobilink()**
Member of **DbmsyncCOM.Dbmsync**

Remarks

Use this event to add custom actions immediately before the remote database connects to the MobiLink server. At this stage, dbmsync has generated the upload.

The ConnectMobiLink event occurs after the BeginSynchronization event.

Example

The following Visual Basic .NET example outputs a message when the ConnectMobilink event is triggered.

```
Private Sub dbmsync1_ConnectMobilink()  
Handles dbmsync1.ConnectMobilink  
  
    MsgBox("Connecting to the MobiLink server")  
  
End Sub
```

DisconnectMobilink event (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

The DisconnectMobilink event is triggered immediately after the component disconnects from the MobiLink server.

Syntax

Public Event **DisconnectMobilink**()
Member of **DbmsyncCOM.Dbmsync**

Remarks

Use this event to add custom actions immediately after the remote database disconnects from the MobiLink server.

Example

The following Visual Basic .NET example outputs a message when the DisconnectMobilink event is triggered.

```
Private Sub dbmsync1_DisconnectMobilink()  
Handles dbmsync1.DisconnectMobilink  
  
    MsgBox("Disconnected from the MobiLink server")  
  
End Sub
```

DoneExecution event (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

The DoneExecution event is triggered when all synchronizations started by a Run method invocation have completed.

Syntax

Public Event **DoneExecution**()
Member of **DbmsyncCOM.Dbmsync**

Remarks

Use this event to add custom actions when all synchronizations started by a Run method invocation have completed.

Example

Using the ExitCode property, the following Visual Basic .NET example outputs the exit code from the synchronizations started by the last Run method invocation:

```
Private Sub dbmsync1_DoneExecution()  
Handles dbmsync1.DoneExecution  
  
    MsgBox(dbmsync1.ExitCode)  
  
End Sub
```

DownloadRow event (removed)

Note

The Dbmlsync integration component has been removed in version 12. In its place, use the dbmlsync programming interface. See [“Dbmlsync API” on page 90](#).

The DownloadRow event is triggered when a row is downloaded from the MobiLink server.

Syntax

```
Public Event DownloadRow(  
    ByVal rowData As DbmlsyncCOM.IRowTransferData  
)  
Member of DbmlsyncCOM.Dbmlsync
```

Parameters

rowData An IRowTransferData object containing details about the downloaded row.

For more information about the IRowTransferData interface, see [“IRowTransferData interface \(removed\)” on page 300](#).

Remarks

Use this event to examine rows being downloaded from the MobiLink server.

To enable the DownloadRow event, use the DownloadEventsEnabled property.

See [“DownloadEventsEnabled property \(removed\)” on page 280](#).

When a delete operation is encountered in the download row event, only primary key column values are available.

Example

The following Visual Basic .NET example iterates through all the columns for a row in the DownloadRow event. It determines if a column value is null, and outputs column names and values.

```
Private Sub dbmlsync1_DownloadRow(  
    ByVal rowData As DbmlsyncCOM.IRowTransferData  
)  
    Handles dbmlsync1.DownloadRow  
  
    Dim liX As Integer  
    For liX = 0 To rowData.ColumnCount - 1  
        If VarType(rowData.ColumnValue(liX)) <> VariantType.Null Then  
            ' output the non-null column value  
            MsgBox("Column " + CStr(liX) + ": " + rowData.ColumnName(liX) + _  
                ", " + CStr(rowData.ColumnValue(liX)))  
        Else  
            ' output 'NULL' for the column value  
            MsgBox("Column " + CStr(liX) + ": " + rowData.ColumnName(liX) + _  
                ", " + "NULL")  
        End If  
    Next liX  
  
End Sub
```

EndDownload event (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

The EndDownload event is triggered at the end of the download stage of the synchronization process.

Syntax

```
Public Event EndDownload(
    long upsertRows,
    long deleteRows
)
Member of DbmsyncCOM.Dbmsync
```

Parameters

upsertRows Indicates the number of rows updated or inserted by the download.

deleteRows Indicates the number of rows deleted by the download.

Remarks

Use this event to add custom actions at the end of the download stage of synchronization.

Example

The following Visual Basic .NET example outputs a message and the number of inserted, updated, and deleted rows when the EndDownload event is triggered.

```
Private Sub dbmsync1_EndDownload(
    ByVal upsertRows As Integer,
    ByVal deleteRows As Integer
)
    Handles dbmsync1.EndDownload

    MsgBox("Download complete." + _
        CStr(upsertRows) + "Rows updated or inserted" + _
        CStr(deleteRows) + "Rows deleted")

End Sub
```

EndLogScan event (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

The EndLogScan event is triggered immediately after the transaction log is scanned for upload. This event is not fired for scripted uploads.

Syntax

Public Event **EndLogScan**()
Member of **DbmlsyncCOM.Dbmlsync**

Remarks

Use this event to add custom actions immediately after the transaction log is scanned for upload.

Example

The following Visual Basic .NET example outputs a message when the EndLogScan event is triggered.

```
Private Sub dbmlsync1_EndLogScan()  
Handles dbmlsync1.EndLogScan  
  
    MsgBox("Scan of transaction log complete...")  
  
End Sub
```

EndSynchronization event (removed)

Note

The Dbmlsync integration component has been removed in version 12. In its place, use the dbmlsync programming interface. See [“Dbmlsync API” on page 90](#).

The EndSynchronization event is triggered when a synchronization is complete.

Syntax

Public Event **EndSynchronization**(
ByVal *exitCode* As Integer,
ByRef *restart* As Boolean
)
Member of **DbmlsyncCOM.Dbmlsync**

Parameters

exitCode If set to anything other than zero, this indicates that a synchronization error occurred.

restart This value is set to false when the event is called. If the event changes its value to true, dbmlsync restarts the synchronization.

Remarks

Use this event to add custom actions when a synchronization is complete.

Example

The following Visual Basic .NET example uses the EndSynchronization event to restart up to five failed synchronization attempts. If all restart attempts failed, the message "All restart attempts failed" is output, along with the exit code. If a synchronization is successful, the message "Synchronization succeeded " is output, along with the exit code.

```

' Global variable for the number of restarts
Dim numberOfRestarts As Integer

Private Sub dbmsync1_EndSynchronization(
    ByVal ExitCode As Integer,
    ByRef restart As Boolean
)
Handles dbmsync1.EndSynchronization

    If numberOfRestarts < 5 Then
        MsgBox("Restart Number: " + CStr(numberOfRestarts + 1))
        If ExitCode <> 0 Then
            ' restart the failed synchronization
            restart = True
            numberOfRestarts = numberOfRestarts + 1
        Else
            ' the last synchronization succeeded
            MsgBox("Synchronization succeeded. " + _
                "Exit code: " + CStr(ExitCode))
        End If
    Else
        MsgBox("All restart attempts failed. " + _
            "Exit code: " + CStr(ExitCode))
    End If

End Sub

```

EndUpload event (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

The EndUpload event is triggered immediately after transmission of the upload to the MobiLink server.

Syntax

Public Event **EndUpload()**
Member of **DbmsyncCOM.Dbmsync**

Remarks

Use this event to add custom actions immediately after transmission of the upload from dbmsync to the MobiLink server.

Example

The following Visual Basic .NET example outputs a message when the EndUpload event is triggered.

```

Private Sub dbmsync1_EndUpload()
Handles dbmsync1.EndUpload

    MsgBox("End Upload")

End Sub

```

Message event (removed)

Note

The Dbmlsync integration component has been removed in version 12. In its place, use the dbmlsync programming interface. See [“Dbmlsync API” on page 90](#).

The Message event is triggered when dbmlsync logs information.

Syntax

```
Public Event Message(_  
    ByVal msgClass As DbmlsyncCOM.MessageClass, _  
    ByVal msgID As Integer, ByVal msg As String_  
)  
Member of DbmlsyncCOM.Dbmlsync
```

Parameters

msgClass indicates the severity of the message. Values can be:

- **MsgInfo** A message containing progress information about the synchronization.
- **MsgDetailedInfo** Like MsgInfo, but containing more details.
- **MsgWarning** A message indicating a potential problem but one that does not prevent successful synchronization.
- **MsgError** A message indicating a problem that prevents successful synchronization.

msgID A unique identifier for the message. If msgID is zero, the message does not have a unique identifier.

msg The text of the message.

Remarks

Use this event to receive information logged by dbmlsync. If you want to add special processing when a specific message is generated, check for it by MsgID. That way, your code continues to work if the text of the message changes.

Example

The following Visual Basic .NET example adds messages logged by dbmlsync to a listbox control.

```
Private Sub dbmlsync1_Message(  
    ByVal msgClass As DbmlsyncCOM.MessageClass,  
    ByVal msgId As Integer, ByVal msg As String  
)  
    Handles dbmlsync1.Message  
    Select Case msgClass  
        Case DbmlsyncCOM.MessageClass.MsgError  
            lstMessages.Items.Add("Error: " + msg)  
        Case DbmlsyncCOM.MessageClass.MsgWarning  
            lstMessages.Items.Add("Warning: " + msg)
```



```

    Case DbmsyncCOM.MessageClass.MsgInfo
        lstMessages.Items.Add("Info: " + msg)
    Case DbmsyncCOM.MessageClass.MsgDetailedInfo
        lstMessages.Items.Add("DetInfo: " + msg)
    End Select
End Sub

```

Example

The following Visual Basic .NET example sets up the Message event to handle errors. Error messages are added to a ListBox control called lstMessages.

```

Private Sub dbmsync1_Message(ByVal msgClass As DbmsyncCOM.MessageClass,
    ByVal msgId As Integer, ByVal msg As String) Handles dbmsync1.Message
    If msgClass = DbmsyncCOM.MessageClass.MsgError Then
        lstMessages.Items.Add("Error: " + msgId.ToString() + " " + msg)
    End If
End Sub

```

To see possible error id values, test run the Dbmsync integration component. For example, if dbmsync returns the error "Unable to connect to MobiLink server", the Message event inserts the following entry in lstMessages:

```

Error: 14173 Unable to connect to MobiLink server.

```

Now, you can associate the error "Unable to connect to MobiLink server" with the error id 14173. The following example sets up the Dbmsync integration component to retry a synchronization whenever error 14173 occurs. The Message event sets a variable called restartSynchronization and resets a variable called numberOfRestarts in response to error 14173. The EndSynchronization event retries the synchronization up to five times.

```

' variables for restarting synchronization
Dim numberOfRestarts As Integer = 0
Dim restartSynchronization As Integer = 0

Private Sub dbmsync1_Message
(
    ByVal msgClass As DbmsyncCOM.MessageClass,
    ByVal msgId As Integer, ByVal msg As String ) Handles dbmsync1.Message

    If msgClass = DbmsyncCOM.MessageClass.MsgError Then
        lstMessages.Items.Add("Error: " + msgId.ToString() + " " + msg)

        If msgId = 14173 Then
            restartSynchronization = 1
            numberOfRestarts = 0
        End If
    End If
End Sub

Private Sub dbmsync1_EndSynchronization(ByVal ExitCode As Integer, _
    ByRef restart As Boolean _
    ) Handles dbmsync1.EndSynchronization

    If restartSynchronization = 1 Then
        If numberOfRestarts < 5 Then

```

```
        restart = True
        numberOfRestarts = numberOfRestarts + 1
    End If
End If
End Sub
```

ProgressIndex event (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

The ProgressIndex event is triggered when dbmsync updates its progress bar.

Syntax

```
Public Event ProgressIndex(_
    ByVal index As Integer, _
    ByVal max As Integer _
)
Member of DbmsyncCOM.Dbmsync
```

Parameters

index An integer representing the progress of the synchronization.

max The maximum progress value. The percentage done = $\text{index}/\text{max} \times 100$. If this value is zero, the maximum value has not changed since the last time the event was fired.

Remarks

Use this event to update a progress indicator such as a progress bar.

Example

The following Visual Basic .NET example updates a progress bar control based on the Index value. The maximum index value is set at the beginning of the synchronization.

```
Private Sub dbmsync1_ProgressIndex(
    ByVal index As Integer,
    ByVal max As Integer
)
    Handles dbmsync1.ProgressIndex

    If max <> 0 Then
        ProgressBar1.Maximum = max
    End If
    ProgressBar1.Value = index

End Sub
```

ProgressMessage event (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

The ProgressMessage event is triggered when synchronization progress information changes.

Syntax

Public Event **ProgressMessage**(ByVal *msg* As String)
Member of **DbmsyncCOM.Dbmsync**

Parameters

msg The new progress string.

Remarks

Use this event to receive the string normally displayed with the dbmsync progress bar.

Example

The following Visual Basic .NET example sets the value of a progress label when the ProgressMessage event is triggered.

```
Private Sub dbmsync1_ProgressMessage(  
    ByVal msg As String  
)  
    Handles dbmsync1.ProgressMessage  
    lblProgressMessage.Text = msg  
End Sub
```

SetTitle event (removed)**Note**

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

The SetTitle event is triggered when status information changes. In the dbmsync utility, this information is displayed in the title bar.

Syntax

Public Event **SetTitle**(ByVal *title*) As String
)
Member of **DbmsyncCOM.Dbmsync**

Parameters

title The title in the dbmsync window title bar.

Remarks

Use this event to receive the title normally seen on the dbmsync window when its value changes.

Example

The following Visual Basic .NET example sets the title of a Windows form when the SetTitle event is triggered.

```
Private Sub dbmsync1_SetTitle(  
    ByVal title As String  
)  
    Handles dbmsync1.SetTitle  
  
    Me.Text = title  
  
End Sub
```

UploadAck event (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

The UploadAck event is triggered after the component has received acknowledgement of the upload from the MobiLink server.

Syntax

```
Public Event UploadAck( _  
    ByVal status As DbmsyncCOM.UploadAckStatus _  
)  
Member of DbmsyncCOM.Dbmsync
```

Parameters

status Indicates the status returned by MobiLink to the remote after the upload is processed. Its value is one of:

- **StatCommitted** Indicates that the upload was received by the MobiLink server and committed.
- **StatRetry** Indicates that the MobiLink server and the remote database had different values for the log offset from which the upload should start. The upload was not committed by the MobiLink server. The component attempts to send another upload starting from the MobiLink server's log offset.
- **StatFailed** Indicates that the MobiLink server did not commit the upload.

Remarks

Use this event to add custom actions after dbmsync has received acknowledgement of the upload from the MobiLink server.

Example

The following Visual Basic .NET example outputs a message if the upload has failed when the UploadAck event is triggered.

```
Private Sub dbmsync1_UploadAck(ByVal status As DbmsyncCOM.UploadAckStatus)
    Handles dbmsync1.UploadAck

    If status = DbmsyncCOM.UploadAckStatus.StatFailed Then
        MsgBox("Upload Failed")
    End If

End Sub
```

UploadRow event (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

The UploadRow event is triggered when a row is uploaded to the MobiLink server.

Syntax

```
Public Event UploadRow(
    ByVal rowData As DbmsyncCOM.IRowTransferData
)
Member of DbmsyncCOM.Dbmsync
```

Parameters

rowData An IRowTransferData object containing details about the uploaded row.

See [“IRowTransferData interface \(removed\)” on page 300](#).

Remarks

Use this event to examine rows being uploaded to the MobiLink server.

To enable the UploadRow event, use the UploadEventsEnabled property. See [“UploadEventsEnabled property \(removed\)” on page 280](#).

Example

The following Visual Basic .NET example iterates through all the columns for a row in the UploadRow event. It determines if a column value is null and outputs column names and values.

```
Private Sub dbmsync1_UploadRow(
    ByVal rowData As DbmsyncCOM.IRowTransferData
)
    Handles dbmsync1.UploadRow

    Dim liX As Integer
    For liX = 0 To rowData.ColumnCount - 1
        If VarType(rowData.ColumnValue(liX)) <> VariantType.Null Then
```

```
        ' output the non-null column value
        MsgBox("Column " + CStr(liX) + ": " + rowData.ColumnName(liX) + _
            ", " + CStr(rowData.ColumnValue(liX)))
    Else
        ' output 'NULL' for the column value
        MsgBox("Column " + CStr(liX) + ": " + rowData.ColumnName(liX) + _
            ", " + "NULL")
    End If
Next liX

End Sub
```

WaitingForUploadAck event (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

The WaitingForUploadAck event is triggered when the component begins waiting for upload acknowledgement from the MobiLink server.

Syntax

Public Event **WaitingForUploadAck()**
Member of **DbmsyncCOM.Dbmsync**

Remarks

Use this event to add custom actions when the component is waiting for upload acknowledgement from the MobiLink server.

Example

The following Visual Basic .NET example outputs a message when the WaitingForUploadAck event is triggered.

```
Private Sub dbmsync1_WaitingForUploadAck()  
Handles dbmsync1.WaitingForUploadAck  
  
    MsgBox("Waiting for Upload Acknowledgement")  
  
End Sub
```

IRowTransferData interface (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

Public Interface **IRowTransferData**
Member of **DbmsyncCOM**

The UploadRow and DownloadRow events accept DbmsyncCOM.IRowTransferData objects as parameters to examine uploaded and downloaded rows. This interface defines detailed row information including the table name, row operation, and column names.

RowOperation property (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

Specifies the operation performed on the row.

Syntax

Public Property **RowOperation()** As DbmsyncCOM.RowEventOp
Member of **DbmsyncCOM.IRowTransferData**

Remarks

This property has one of the following values:

OpInsert The row was inserted.

OpUpdate The row was updated.

OpDelete The row was deleted.

OpTruncate The table was truncated (all the rows in the table were deleted). When the RowOperation property has this value, the ColumnName and ColumnValue properties return invalid information.

Note: For the DownloadRow event, upsert (update or insert) operations are given the OpInsert value.

TableName property (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

The name of the table on which an upload or download operation occurred.

Syntax

Public Property **TableName()** As String
Member of **DbmsyncCOM.IRowTransferData**

Remarks

The TableName property specifies the name of the table on which an upload or download operation occurred. The following example illustrates the use of the TableName property in the UploadRow event.

See [“UploadRow event \(removed\)” on page 299](#).

Example

The following is a Visual Basic .NET example.

```
Private Sub dbmsync1_UploadRow(  
    ByVal rowData As DbmsyncCOM.IRowTransferData  
)  
    Handles dbmsync1.UploadRow  
    MsgBox ("Table name:" + rowData.TableName)  
End Sub
```

ColumnName property (removed)

Note

The Dbmsync integration component has been removed in version 12. In its place, use the dbmsync programming interface. See [“Dbmsync API” on page 90](#).

Retrieves the column names for a row on which an upload or download operation occurred.

Syntax

Public Property **ColumnName**(ByVal *index* As Integer) As Object
Member of **DbmsyncCOM.IRowTransferData**

Parameters

index A zero based integer specifying the column name to be retrieved. Index values range from zero to one less than the ColumnCount property value.

See [“ColumnCount property \(removed\)” on page 304](#).

Remarks

Associated column values can be retrieved using the ColumnValue property with the same index.

Example

The following Visual Basic .NET example iterates through all the columns for a row in the UploadRow event. It determines if a column value is null and outputs column names and values.

See [“UploadRow event \(removed\)” on page 299](#).

```
Private Sub dbmsync1_UploadRow(  
    ByVal rowData As DbmsyncCOM.IRowTransferData  
)  
    Handles dbmsync1.UploadRow  
  
    Dim liX As Integer  
    For liX = 0 To rowData.ColumnCount - 1  
        If VarType(rowData.ColumnValue(liX)) <> VariantType.Null Then  
            ' output the non-null column value
```



```

        MsgBox("Column " + CStr(liX) + ": " + rowData.ColumnName(liX) + _
            ", " + CStr(rowData.ColumnValue(liX)))
    Else
        ' output 'NULL' for the column value
        MsgBox("Column " + CStr(liX) + ": " + rowData.ColumnName(liX) + _
            ", " + "NULL")
    End If
Next liX

End Sub

```

ColumnValue property (removed)

Note

The Dbmlsync integration component has been removed in version 12. In its place, use the dbmlsync programming interface. See [“Dbmlsync API” on page 90](#).

Retrieves the value of columns on which an upload or download operation occurred.

Syntax

Public Property **ColumnValue**(ByVal *index* As Integer) As Object
Member of **DbmlsyncCOM.IRowTransferData**

Parameters

index The zero based integer specifying the column value to be retrieved. Index values range from zero to one less than the ColumnCount property value.

See [“ColumnCount property \(removed\)” on page 304](#).

Remarks

When an update operation is encountered, the column values given by this property are the values after the update is applied.

Associated column names can be retrieved using the ColumnName property with the same index.

BLOB column values are not available through this property. When a BLOB column is encountered, the ColumnValue is the string "(blob)".

Example

The following Visual Basic .NET example iterates through all the columns for a row in the UploadRow event. It determines if a column value is null and outputs column names and values.

See [“UploadRow event \(removed\)” on page 299](#).

```

Private Sub dbmlsync1_UploadRow(
    ByVal rowData As DbmlsyncCOM.IRowTransferData
)
    Handles dbmlsync1.UploadRow

    Dim liX As Integer

```

```
For liX = 0 To rowData.ColumnCount - 1
  If VarType(rowData.ColumnValue(liX)) <> VariantType.Null Then
    ' output the non-null column value
    MsgBox("Column " + CStr(liX) + ": " + rowData.ColumnName(liX) + _
      ", " + CStr(rowData.ColumnValue(liX)))
  Else
    ' output 'NULL' for the column value
    MsgBox("Column " + CStr(liX) + ": " + rowData.ColumnName(liX) + _
      ", " + "NULL")
  End If
Next liX

End Sub
```

ColumnCount property (removed)

Note

The Dbmlsync integration component has been removed in version 12. In its place, use the dbmlsync programming interface. See [“Dbmlsync API” on page 90](#).

The number of columns contained in a row on which an upload or download operation occurred.

Syntax

Public Property **ColumnCount**() As Integer
Member of **DbmlsyncCOM.IRowTransferData**

Remarks

The ColumnCount property specifies the number of columns for a row on which an upload or download operation occurred. The following example illustrates the use of the ColumnCount property in the UploadRow event.

See [“UploadRow event \(removed\)” on page 299](#).

Example

The following is a Visual Basic .NET example.

```
Private Sub dbmlsync1_UploadRow(
  ByVal rowData As DbmlsyncCOM.IRowTransferData
)
  Handles dbmlsync1.UploadRow

  MsgBox "Number of Columns:" + CStr(rowData.ColumnCount)

End Sub
```

DBTools interface for dbmlsync

Introduction to DBTools interface for dbmlsync

Database tools (DBTools) is a library you can use to integrate database management, including synchronization, into your applications. All the database management utilities are built on DBTools.

See [“Database tools interface \(DBTools\)” \[SQL Anywhere Server - Programming\]](#).

Note

The Dbmlsync API is the preferred interface for integrating synchronization into your applications. It provides functionality that is very similar to the DBTools interface and is easier to use.

See [“Dbmlsync API” on page 90](#).

You can use the DBTools interface for dbmlsync to integrate synchronization functionality into your MobiLink synchronization client applications. For example, you can use the interface launch synchronizations and to display dbmlsync output messages in a custom user interface.

The DBTools interface for dbmlsync consists of the following elements that let you configure and run the MobiLink synchronization client:

- **a_sync_db structure** This structure holds settings, corresponding to dbmlsync command line options, that allow you to customize synchronization. This structure also contains pointers to callback functions that receive synchronization and progress information.

See [“a_sync_db structure” \[SQL Anywhere Server - Programming\]](#).

- **a_syncpub structure** This structure holds publication or subscription information. You can specify a linked list of publications or subscriptions for synchronization.

See [“a_syncpub structure” \[SQL Anywhere Server - Programming\]](#).

- **DBSynchronizeLog function** This function starts the synchronization process. Its only parameter is a pointer to an a_sync_db instance.

See [“DBSynchronizeLog function” \[SQL Anywhere Server - Programming\]](#).

Setting up the DBTools interface for dbmlsync

This section guides you through the basic steps for using the DBTools interface for dbmlsync.

For more information about the DBTools library, see [“Database tools interface \(DBTools\)” \[SQL Anywhere Server - Programming\]](#).

For more information about using import libraries for your development environment, see [“Using the database tools interface” \[SQL Anywhere Server - Programming\]](#).

To configure and start dbmlsync using the DBTools interface in C or C++

1. Include the DBTools header file.

The DBTools header file, *dbtools.h*, lists the entry points to the DBTools library and defines required data types.

```
#include "dbtools.h"
```

2. Start the DBTools interface.

- Declare and initialize the `a_dbtools_info` structure.

```
a_dbtools_info  info;
short ret;
...
// clear a_dbtools_info fields
memset( &info, 0, sizeof( info ) );
info.errorrtn = dbsyncErrorCallBack;
```

The `dbsyncErrorCallBack` function handles error messages and is defined in step 4 of this procedure.

- Use the `DBToolsInit` function to initialize DBTools.

```
ret = DBToolsInit( &info );
if( ret != 0 ) {
    printf("dbtools initialization failure \n");
}
```

For more information about DBTools initialization, see:

- “Using the database tools interface” [[SQL Anywhere Server - Programming](#)]
- “`a_dbtools_info` structure” [[SQL Anywhere Server - Programming](#)]
- “`DBToolsInit` function” [[SQL Anywhere Server - Programming](#)]

3. Initialize the `a_sync_db` structure.

- Declare an `a_sync_db` instance. For example, declare an instance called `dbsync_info`:

```
a_sync_db dbsync_info;
```

- Clear `a_sync_db` structure fields.

```
memset( &dbsync_info, 0, sizeof( dbsync_info ) );
```

- Set required `a_sync_db` fields.

```
dbsync_info.version = DB_TOOLS_VERSION_NUMBER;
dbsync_info.output_to_mobile_link = 1;
dbsync_info.default_window_title
    = "dbmlsync dbtools sample";
```

- Set the database connection string.

```
dbsync_info.connectparms = "uid=DBA;pwd=sql";
```

For more information about database connection parameters, see “`-c dbmlsync option`” on page 101.

- Set other `a_sync_db` fields to customize synchronization.

Most fields correspond to dbmlsync command line options. For more information about this correspondence, see *dbtools.h*.

In the example below, verbose operation is enabled.

```
dbsync_info.verbose_upload = 1;
dbsync_info.verbose_option_info = 1;
dbsync_info.verbose_row_data = 1;
dbsync_info.verbose_row_cnts = 1;
```

For more information about a_sync_db fields, see “[a_sync_db structure](#)” [*SQL Anywhere Server - Programming*].

4. Create callback functions to receive feedback during synchronization and assign these functions to the appropriate a_sync_db fields.

The following functions use the standard output stream to display dbmlsync error, log, and progress information.

For more information about DBTools callback functions, see “[Using callback functions](#)” [*SQL Anywhere Server - Programming*].

- For example, create a function called `dbsyncErrorCallBack` to handle generated error messages:

```
extern short _callback dbsyncErrorCallBack( char *str )
{
    if( str != NULL ) {
        printf( "Error Msg    %s\n", str );
    }
    return 0;
}
```

- For example, create a function called `dbsyncWarningCallBack` to handle generated warning messages:

```
extern short _callback dbsyncWarningCallBack( char *str )
{
    if( str != NULL ) {
        printf( "Warning Msg  %s\n", str );
    }
    return 0;
}
```

- For example, create a function called `dbsyncLogCallBack` to receive verbose informational messages that you might choose to log to a file instead of displaying in a window:

```
extern short _callback dbsyncLogCallBack( char *str )
{
    if( str != NULL ) {
        printf( "Log Msg      %s\n", str );
    }
    return 0;
}
```

- For example, create a function called `dbsyncMsgCallBack` to receive informational messages generated during synchronization.

```
extern short _callback dbsyncMsgCallBack( char *str )
{
    if( str != NULL ) {
```

```

        printf( "Display Msg  %s\n", str );
    }
    return 0;
}

```

- For example, create a function called `dbsyncProgressMessageCallBack` to receive the progress text. In the `dbmlsync` utility, this text is displayed directly above the progress bar.

```

extern short _callback dbsyncProgressMessageCallBack(
    char *str )
{
    if( str != NULL ) {
        printf( "ProgressText %s\n", str );
    }
    return 0;
}

```

- For example, create a function called `dbsyncProgressIndexCallBack` to receive information for updating a progress indicator or progress bar. This function receives two parameters:
 - **index** An integer representing the current progress of a synchronization.
 - **max** The maximum progress value. If this value is zero, the maximum value has not changed since the last time the event was fired.

```

extern short _callback dbsyncProgressIndexCallBack
(a_sql_uint32 index, a_sql_uint32 max )
{
    printf( "ProgressIndex    Index %d Max: %d\n",
        index, max );
    return 0;
}

```

A typical sequence of calls to this callback is shown below

```

// example calling sequence
dbsyncProgressIndexCallBack( 0, 100 );
dbsyncProgressIndexCallBack( 25, 0 );
dbsyncProgressIndexCallBack( 50, 0 );
dbsyncProgressIndexCallBack( 75, 0 );
dbsyncProgressIndexCallBack( 100, 0 );

```

This sequence should result in the progress bar being set to 0% done, 25% done, 50% done, 75% done, and 100% done.

- For example, create a function called `dbsyncWindowTitleCallBack` to receive status information. In the `dbmlsync` utility, this information is displayed in the title bar.

```

extern short _callback dbsyncWindowTitleCallBack(
    char *title )
{
    printf( "Window Title    %s\n", title );
    return 0;
}

```

- The `dbsyncMsgQueueCallBack` function is called when a delay or sleep is required. It must return one of the following values, which are defined in `dllapi.h`.
 - **MSGQ_SLEEP_THROUGH** indicates that the routine slept for the requested number of milliseconds.

- **MSGQ_SHUTDOWN_REQUESTED** indicates that you would like the synchronization to terminate as soon as possible.
- **MSGQ_SYNC_REQUESTED** indicates that the routine slept for less than the requested number of milliseconds and that the next synchronization should begin immediately if a synchronization is not currently in progress.

```
extern short _callback dbsyncMsgQueueCallBack(
    a_sql_uint32 sleep_period_in_milliseconds )
{

    printf( "Sleep %d ms\n", sleep_period_in_milliseconds );
    Sleep( sleep_period_in_milliseconds );
    return MSGQ_SLEEP_THROUGH;
}
```

- Assign callback function pointers to the appropriate `a_sync_db` synchronization structure fields.

```
// set call back functions
dbsync_info.errorrtn      = dbsyncErrorCallBack;
dbsync_info.warningrtn   = dbsyncWarningCallBack;
dbsync_info.logrtn       = dbsyncLogCallBack;
dbsync_info.msgrtn       = dbsyncMsgCallBack;
dbsync_info.msgqueue rtn = dbsyncMsgQueueCallBack;
dbsync_info.progress_index_rtn
    = dbsyncProgressIndexCallBack;
dbsync_info.progress_msg_rtn
    = dbsyncProgressMessageCallBack;
dbsync_info.set_window_title_rtn
    = dbsyncWindowTitleCallBack;
```

5. Create a linked list of `a_syncpub` structures to specify which subscriptions should be synchronized.

Each node in the linked list corresponds to one instance of the `-s` option on the `dbmsync` command line.

- Declare an `a_syncpub` instance. For example, call it `publication_info`:

```
a_syncpub publication_info;
```

- Initialize `a_syncpub` fields, specifying subscriptions you want to synchronize.

For example, to synchronize the `template_p1` and `template_p2` subscriptions together in a single synchronization session:

```
publication_info.next = NULL; // linked list terminates
publication_info.subscription = "template_p1,template_p2";
publication_info.ext_opt     = "dir=c:\\logs";
publication_info.allocated_by_dbsync = 0;
publication_info.pub_name   = NULL;
```

This is equivalent to specifying `-s template_p1,template_p2` on the `dbmsync` command line.

Specifying extended options using the `ext_opt` field, provides the same functionality as the `dbmsync -eu` option.

See “[-eu dbmsync option](#)” on page 108.

- Assign the publication structure to the `upload_defs` field of your `a_sync_db` instance.

```
dbsync_info.upload_defs = &publication_info;
```

You can create a linked list of `a_syncpub` structures. Each `a_syncpub` instance in the linked list is equivalent to one specification of the `-n` or `-s` option on the `dbmsync` command line.

See: and .

- “`-s dbmsync option`” on page 117
- “`-n dbmsync option (deprecated)`” on page 110
- “`a_syncpub structure`” [*SQL Anywhere Server - Programming*]

6. Run `dbmsync` using the `DBSynchronizeLog` function.

In the following code listing, `sync_ret_val` contains the return value 0 for success or non-0 for failure.

```
short sync_ret_val;
printf("Running dbmsync using dbtools interface...\n");
sync_ret_val = DBSynchronizeLog(&dbsync_info);
printf("\n Done... synchronization return value is: %I \n", sync_ret_val);
```

You can repeat step 6 multiple times with the same or different parameter values.

7. Shutdown the DBTools interface.

The `DBToolsFini` function frees DBTools resources.

```
DBToolsFini( &info );
```

See “`DBToolsFini function`” [*SQL Anywhere Server - Programming*].

Scripted upload

Introduction to scripted upload

Scripted upload applies only to MobiLink applications that use SQL Anywhere remote databases.

Warning

When you implement scripted upload, `dbmsync` does not use the transaction log to determine what to upload. As a result, if your scripts do not capture all changes, data on remote databases can be lost. For these reasons, log-based synchronization is the recommended synchronization method for most applications.

In most MobiLink applications, the upload is determined by the database transaction log so that changes made to the remote database since the last upload are synchronized. This is the appropriate design for most applications and ensures that data on the remote is not lost.

However, in some cases you may want to ignore the transaction log and define the upload yourself. Using scripted upload you can define exactly what data you want to upload. When doing scripted upload you do not have to maintain a transaction log for your remote database. Transaction logs take up space that may be at a premium on small devices. However, transaction logs are very important for database backup and recovery, and improve database performance.

To implement scripted upload, you create a special kind of publication that specifies the names of stored procedures that you create. The stored procedures define an upload by returning result sets that contain the rows to insert, update, or delete on the consolidated database.

Note: Do not confuse scripted upload with upload scripts. Upload scripts are MobiLink event scripts on the consolidated database that you write to tell the MobiLink server what to do with the upload. When you use scripted upload, you still need to write upload scripts to apply uploads to the consolidated database and download scripts to determine what to download.

Scenarios

The following are some scenarios where scripted upload may be useful:

- Your remote database is running on a device with limited storage and there is not enough space for a transaction log.
- You want to upload all the data from all your remote databases to create a new consolidated database.
- You want to write custom logic to determine which changes are uploaded to the consolidated database.

Warnings

Before implementing scripted upload, be sure to read this entire section. In particular, take note of the following points:

- If you do not set up your scripted upload correctly, you can lose data.
- When you implement scripted upload, you need to maintain or reference things that dbmlsync normally handles for you. These include pre- and post-images of data, and the progress of the synchronization.
- You need to lock tables on the remote database while synchronizing via scripted upload. With log-based synchronization, locking is not required.
- Transactional uploads are extremely difficult to implement with scripted upload.

Setting up scripted upload

The following steps provide an overview of the tasks required to set up scripted upload, assuming that you already have MobiLink synchronization set up.

Overview of setting up scripted upload

1. Create stored procedures that identify the rows to upload. You can define three stored procedures per table: one each for upload, insert, and delete.

See [“Defining stored procedures for scripted upload” on page 318](#).
2. Create a publication that contains the keywords WITH SCRIPTED UPLOAD and that specifies the names of the stored procedures.

See [“Creating publications for scripted upload” on page 322](#).

Caution

When using scripted upload, it is strongly recommended that you use the default setting for the dbmsync extended option LockTables.

You can avoid many problems with scripted uploads by using the default setting for LockTables, which causes dbmsync to obtain locks on all synchronization tables before the upload is built. This prevents other connections from changing the synchronization tables while your scripts are building the upload. It also ensures that there are no uncommitted transactions that affect synchronization tables open while your scripts are building the upload.

Other resources for getting started

- [“Scripted upload example” on page 323](#)

Design considerations for scripted upload

One operation per row

The upload may not contain more than one operation (insert, update, or delete) for a single row. However, you can combine multiple operations into a single upload operation; for example, if a row is inserted and then updated you can replace the two operations with a single insert of the final values.

Order of operations

When the upload is applied to the consolidated database, insert and update operations are applied before delete operations. You cannot make any other assumptions about the order of operations within a given table.

Handling conflicts

A conflict occurs when a row is updated on more than one database between synchronizations. The MobiLink server can identify conflicts because each update operation in an upload contains the pre-image of the row being updated. The pre-image is the value of all the columns in the row the last time it was successfully uploaded or downloaded. The MobiLink server identifies a conflict when the pre-image does not match the values in the consolidated database when the upload is applied.

If your application needs conflict detection and you are using scripted upload, then on the remote database you need to keep track of the value of each row the last time it was successfully uploaded or downloaded. This allows you to upload the correct pre-images.

One way to maintain pre-image data is to create a pre-image table that is identical to your synchronization table. You can then create a trigger on your synchronization table that populates the pre-image table each time an update executes. After a successful upload you can delete the rows in the pre-image table.

For an example that implements conflict resolution, see [“Scripted upload example” on page 323](#).

Not handling conflicts

If you do not need to handle conflict detection, you can simplify your application considerably by not tracking pre-images. Instead, you upload updates as insert operations. You can then write an `upload_insert` script on the consolidated database that inserts a row if it does not already exist or updates the row if it does exist. If you are using a SQL Anywhere consolidated database, you can achieve this with the `ON EXISTING` clause in the `INSERT` statement in your `upload_insert` script.

See “[INSERT statement](#)” [*SQL Anywhere Server - SQL Reference*].

When you do not handle conflicts and two or more remote databases change the same row, the last one to synchronize overrides the earlier changes.

Handling forced conflicts

For delete operations, it is essential that the primary key of a row that is uploaded is correct. However, it does not matter if the values of the non-primary key columns match those in the consolidated database. The only case where the value of non-primary key columns is important is when forced conflict mode is used at the MobiLink server. In that case, all the column values are passed to the `upload_old_row_insert` script on the consolidated database. Depending on how you have implemented this script, it may be necessary for non-primary key column values to be correct.

See “[Forced conflicts \(Deprecated\)](#)” [*MobiLink - Server Administration*].

Locking

You can avoid many problems with scripted uploads by using the default setting for the `dbmsync` extended option `LockTables`, which causes `dbmsync` to obtain exclusive locks on all synchronization tables before the upload is built. This prevents other connections from changing the synchronization tables while your scripts are building the upload. It also ensures that there are no uncommitted transactions that affect synchronization tables open while your scripts are building the upload.

If you must turn off table locking, see “[Scripted upload with no table locking](#)” on page 315.

Redundant uploads

Typically, you want to upload each operation on the remote database exactly once. To help you with this, MobiLink maintains a progress value for each subscription. By default the progress value is the time at which `dbmsync` began building the last successful upload. This progress value can be overridden with a different value using the `sp_hook_dbmsync_set_upload_end_progress` hook.

See “[sp_hook_dbmsync_set_upload_end_progress](#)” on page 222.

Each time one of your upload procedures is called, values are passed to it through the `#hook_dict` table. Among these are the 'start progress' and 'end progress' values. These define the period of time for which the upload being built should include changes to the remote database. Operations that occurred before the 'start progress' have already been uploaded. Those that occur after the 'end progress' should be uploaded during the next synchronization.

Unknown Upload Status

A common mistake in the implementation of scripted upload is creating stored procedures that depend on knowing whether an upload was successfully applied to the consolidated database in the `sp_hook_dbmlsync_upload_end` or `sp_hook_dbmlsync_end` hooks. This approach is unreliable.

For example, the following example tries to handle inserts by using a bit on each row to keep track of whether the row needs to be uploaded. The bit is set when a row is inserted, and it is cleared in the `sp_hook_dbmlsync_upload_end` hook when the upload is successfully committed.

```
//
// DO NOT DO THIS!
//
CREATE TABLE t1 (
    pk          integer primary key,
    val         varchar( 256 ),
    to_upload   bit DEFAULT 1
);

CREATE PROCEDURE t1_ins()
RESULT( pk integer, val varchar(256) )
BEGIN
    SELECT pk, val
    FROM t1
    WHERE to_upload = 1;
END;

CREATE PROCEDURE sp_hook_dbmlsync_upload_end()
BEGIN
    DECLARE    upload_status  varchar(256);

    SELECT value
    INTO upload_status
    FROM #hook_dict
    WHERE name = 'upload status';

    if upload_status = 'committed' THEN
        UPDATE t1 SET to_upload = 0;
    END IF
END;

CREATE PUBLICATION p1 WITH SCRIPTED UPLOAD (
    TABLE t1 USING ( PROCEDURE t1_ins FOR UPLOAD INSERT )
);
```

This approach works most of the time. It fails when a hardware or software failure occurs that stops `dbmlsync` after the upload has been sent but before it has been acknowledged by the server. In that case, the upload may be applied to the consolidated database but the `sp_hook_dbmlsync_upload_end` hook is not called and the `to_upload` bits are not cleared. As a result, in the next synchronization, inserts are uploaded for rows that have already been uploaded. Usually this causes the synchronization to fail because it generates a duplicate primary key error on the consolidated database.

The other case where problems can occur is when communication with the MobiLink server is lost after the upload is sent but before it has been acknowledged. In this case `dbmlsync` cannot tell if the upload was successfully applied. `Dbmlsync` calls the `sp_hook_dbmlsync_upload_end` hook and sets the upload status to unknown. As the hook is written this prevents it from clearing the `to_upload` bits. If the upload was not applied by the server, this is correct. However, if the upload was applied then the same problem occurs as

in the previous paragraph. In both of these cases, the affected remote database is unable to synchronize again until someone manually intervenes to resolve the problem.

Preventing data loss during download

When using scripted uploads, it is possible for data in the remote database that needs to be uploaded to be overwritten by data being downloaded from the consolidated database. This results in the loss of changes made to the remote database. To prevent this data loss your upload procedures must include all changes that were committed in the remote database before the `sp_hook_dbmsync_set_upload_end_progress` hook was called in each upload they build.

The following example shows how data can be lost if you violate this rule:

Time	
1:05:00	A row, R, that exists both in the consolidated and remote databases is updated with some new values, R1, in the remote database and the change is committed.
1:06:00	The row R is updated in the consolidated database to some new values R2 and the change is committed.
1:07:00	A synchronization occurs. The upload scripts are written so that the upload only contains operations committed before 1:00:00. This violates our rule because it prevents all operations that occurred before the upload was built from being uploaded. The change to row R is not included upload because it occurred after 1:00:00. The download received from the server contains the row R2. When the download is applied, the row R2 replaces the row R1 in the remote database. The update on the remote database is lost.

Scripted upload with no table locking

By default, `dbmsync` locks the tables being synchronized before any upload scripts are called, and it maintains these locks until the download is committed. You can prevent table locking by setting the extended option `LockTables` to off.

When possible, it is recommended that you use the default table locking behavior. Doing scripted uploads without table locking significantly increases the number of issues you must consider and the difficulty of creating a correct and workable solution. This should only be attempted by advanced users with a good understanding of database concurrency and synchronization concepts.

Using isolation levels with no table locks

When table locking is off, the isolation level at which your upload stored procedures run is very important because it determines how uncommitted transactions are handled. This is not an issue when table locking is on because table locks ensure that there are no uncommitted changes on the synchronized tables when the upload is built.

Your upload stored procedures run at the default isolation level for the database user who is specified in the `dbmsync` command line unless you explicitly change the isolation level in your upload stored procedure.

Isolation level 0 is the default isolation level for the database, but it is recommended that you do not run your upload procedures at isolation level 0 when using scripted upload with no table locks. If you implement scripted upload without table locks and use isolation level 0, you may upload changes that are not committed, which could result in the following problems:

- The uncommitted changes could be rolled back, which would result in incorrect data being sent to the consolidated database.
- The uncommitted transaction may not be complete, in which case you might upload only part of a transaction and leave the consolidated database in an inconsistent state.

Your alternatives are to use isolation levels 1, 2, 3, or snapshot. All of these isolation levels ensure that you do not upload uncommitted transactions.

Using isolation levels 1, 2, or 3 could result in your upload stored procedures blocking if there are uncommitted changes on the table. Since your upload stored procedures are called while dbmlsync is connected to the MobiLink server, this could tie up server connections. If you use isolation level 1, you may be able to avoid blocking by using the READPAST table-hint clause in your select statements.

Snapshot isolation is a good choice since it prevents both blocking and reads of uncommitted changes.

Losing Uncommitted Changes

If you choose to forgo table locking, you must have a mechanism for handling operations that are not committed when a synchronization occurs. To see why this is a problem, consider the following example.

Suppose a table is being synchronized by scripted upload. For simplicity, assume that only inserts are being uploaded. The table contains an insert_time column that is a timestamp that indicates the time when each row was inserted.

Each upload is built by selecting all the committed rows in the table whose insert_time is after the last successful upload and before the time when you started to build the current upload (which is the time when the sp_hook_dbmlsync_set_upload_end_progress hook was called). Suppose the following takes place.

Time	
1:00:00	A successful synchronization occurs.
1:04:00	Row R is inserted into the table but not committed. The insert_time column for R is set to 1:04:00.
1:05:00	A synchronization occurs. Rows with insert times between 1:00:00 and 1:05:00 are uploaded. Row R is not uploaded because it is uncommitted. The synchronization progress is set to 1:05:00.
1:07:00	The row inserted at 1:04:00 is committed. The insert_time column for R continues to contain 1:04:00.

Time	
1:10:00	A synchronization occurs. Rows with insert times between 1:05:00 and 1:10:00 are uploaded. Row R is not uploaded because its insert_time is not in the range. In fact, row R is never uploaded.

In general, any operation that occurs before a synchronization but is committed after the synchronization is susceptible to loss in this way.

Handling uncommitted transactions

The simplest way to handle uncommitted transactions is to use the `sp_hook_dbmsync_set_upload_end_progress` hook to set the end progress for each synchronization to the start time of the oldest uncommitted transaction at the time the hook is called. You can determine this time using the `sa_transactions` system procedure as follows:

```
SELECT min( start_time )
FROM sa_transactions()
```

In this case, your upload stored procedures must ignore the end progress that was calculated in the `sp_hook_dbmsync_set_upload_end_progress` hook using `sa_transactions` and passed in using the `#hook_dict` table. The stored procedures should just upload all committed operations that occurred after the start progress. This ensures that the download does not overwrite rows with changes that still need to be uploaded. It also ensures that operations are uploaded in a timely manner even when there are uncommitted transactions.

This solution ensures that no operations are lost, but some operations may be uploaded more than once. Your scripts on the server side must be written to handle operations being uploaded more than once. Below is an example that shows how a row can be uploaded more than once in this setup.

Time	
1:00:00	A successful synchronization occurs.
2:00:00	Row R1 is inserted but not committed.
2:10:00	Row R2 is inserted and committed.
3:00:00	A synchronization occurs. Operations that occurred between 1:00 and 3:00 are uploaded. Row R2 is uploaded and the progress is set to 2:00 because that is the start time of the oldest uncommitted transaction.
4:00:00	Row R1 is committed.
5:00:00	A synchronization occurs. Operations that occurred between 2:00 and 5:00 are uploaded and the progress is set to 5:00. The upload contains rows R1 and R2 because they both have timestamps within the upload range. So, R2 has been uploaded twice.

If your consolidated database is SQL Anywhere, you can handle redundantly uploaded insert operations by using the `INSERT ... ON EXISTING UPDATE` statement in your `upload_insert` script in the consolidated database.

For other consolidated databases, you can implement similar logic in a stored procedure that is called by your `upload_insert` script. Just write a check to see if a row with the primary key of the row being inserted already exists in the consolidated database. If the row exists update it, otherwise insert the new row.

Redundantly uploaded delete and update operations are a problem when you have conflict detection or resolution logic on the server side. If you write conflict detection and resolution scripts on the server side, they must be able to handle redundant uploads.

Redundantly uploaded deletes can be a major concern if primary key values can be reused by the consolidated database. Consider the following sequence of events:

1. Row R with primary key 100 is inserted into a remote database and uploaded to the consolidated database.
2. Row R is deleted on the remote database and the delete operation is uploaded.
3. A new row R' with primary key 100 is inserted into the consolidated database.
4. The delete operation on row R from step 2 is uploaded again from the remote database. This could easily result in R' being deleted inappropriately from the consolidated database.

See also

- [“sa_transactions system procedure” \[SQL Anywhere Server - SQL Reference\]](#)
- [“Set the isolation level” \[SQL Anywhere Server - SQL Usage\]](#)

Defining stored procedures for scripted upload

To implement scripted upload, you create stored procedures that define the upload by returning result sets that contain the rows to update, insert, or delete on the consolidated database.

When the stored procedures are called, a temporary table called `#hook_dict` is created that has two columns: `name` and `value`. The table is used to pass name-value pairs to your stored procedures. Your stored procedures can retrieve useful information from this table.

The following name-value pairs are defined:

Name	Value	Description
start progress	timestamp as string	The time up to which all changes on the remote database have been uploaded. Your upload should only reflect operations that occur after this time.

Name	Value	Description
raw start progress	64-bit unsigned integer	The start progress expressed as an unsigned integer.
end progress	timestamp as string	The end of the upload period. Your upload should only reflect operations that occur before this time.
raw end progress	64-bit unsigned integer	The end progress expressed as an unsigned integer.
generating download exclusion list	true false	True if the synchronization is download-only or file-based. In those cases no upload is sent, and the download is not applied if it affects any row selected by a scripted upload stored procedure. (This ensures that changes made at the remote that need to be uploaded are not overwritten by the download.)
subscription_ <i>n</i>	subscription name(s)	The names of subscriptions being synchronized where <i>n</i> is an integer. This is one subscription_ <i>n</i> entry for each subscription being synchronized. The numbering of <i>n</i> starts at zero.
publication_ <i>n</i>	publication name	Deprecated. Use subscription_ <i>n</i> instead. The publications being synchronized, where <i>n</i> is an integer. The numbering of <i>n</i> starts at zero.
script version	version name	The MobiLink script version to be used for the synchronization.
MobiLink user	MobiLink user name	The MobiLink user for which you are synchronizing.

See “#hook_dict table” on page 180.

Custom progress values in scripted upload

By default, the start progress and end progress values passed to your scripted upload procedures represent timestamps. By default the end progress is the time when dbmlsync starts to build the upload. The start progress for a synchronization is always the end progress used for the most recent successful upload of that subscription. This default behavior is appropriate for most implementations.

The `sp_hook_dbmlsync_set_upload_end_progress` hook is provided for the rare cases where different behavior is required. Using this hook, you can set the end progress to be used for an upload. The end progress you choose must be greater than the start progress. You cannot alter the start progress.

In the `sp_hook_dbmlsync_set_upload_end_progress` hook you can specify the end progress either as a timestamp or as an unsigned integer. The value is available in either form to the upload stored procedures. For your convenience, the `sa_convert_ml_progress_to_timestamp` and `sa_convert_timestamp_to_ml_progress` functions can be used to convert progress values between the two forms.

See:

- “`sp_hook_dbmlsync_set_upload_end_progress`” on page 222
- “`sa_convert_ml_progress_to_timestamp` system procedure” [*SQL Anywhere Server - SQL Reference*]
- “`sa_convert_timestamp_to_ml_progress` system procedure” [*SQL Anywhere Server - SQL Reference*]

Defining stored procedures for inserts

The stored procedures for inserts must return result sets containing all the columns to be uploaded, as defined in the `CREATE PUBLICATION` statement, in the same order that the columns were declared in the `CREATE TABLE` statement.

Column order

You can find the creation order of columns in a table called T1 with the following query:

```
SELECT column_name
FROM SYSTAB JOIN SYSTABCOL
WHERE table_name = 't1'
ORDER BY column_id
```

Example

For a detailed explanation of how to define stored procedures for inserts, see “[Scripted upload example](#)” on page 323.

The following example creates a table called `t1` and a publication called `p1`. The publication specifies `WITH SCRIPTED UPLOAD` and registers the stored procedure `t1_insert` as the insert procedure. In the definition of the `t1_insert` stored procedure, the result set includes all columns listed in the `CREATE PUBLICATION` statement but in the order in which the columns were declared in the `CREATE TABLE` statement.

```
CREATE TABLE t1(
  //The column ordering is taken from here
  pk integer primary key,
  c1 char( 30),
  c2 float,
  c3 double );

CREATE PROCEDURE t1_insert (
  RESULT( pk integer, c1 char(30), c3 double )
begin
  ...
```

```

end

CREATE PUBLICATION WITH SCRIPTED UPLOAD p1(
  // Order of columns here is ignored
  TABLE t1( c3, pk, c1 ) USING (
    PROCEDURE t1_insert FOR UPLOAD INSERT
  )
)

```

Defining stored procedures for deletes

The stored procedures for deletes must return result sets containing all the columns to be uploaded, as defined in the CREATE PUBLICATION statement, in the same order that the columns were declared in the CREATE TABLE statement.

Column order

You can find the creation order of columns in a table called T1 with the following query:

```

SELECT column_name
FROM SYSTAB JOIN SYSTABCOL
  WHERE table_name = 't1'
ORDER BY column_id

```

Example

For a detailed explanation of how to define stored procedures for deletes, see [“Scripted upload example” on page 323](#).

The following example creates a table called t1 and a publication called p1. The publication specifies WITH SCRIPTED UPLOAD and registers the stored procedure t1_delete as the delete procedure. In the definition of the t1_delete stored procedure, the result set includes all columns listed in the CREATE PUBLICATION statement but in the order in which the columns were declared in the CREATE TABLE statement.

```

CREATE TABLE t1(
  //The column ordering is taken from here
  pk integer primary key,
  c1 char( 30),
  c2,    float,
  c3 double );

CREATE PROCEDURE t1_delete ()
RESULT( pk integer, c1 char(30), c3 double )
begin
  ...
end

CREATE PUBLICATION p1 WITH SCRIPTED UPLOAD(
  // Order of columns here is ignored
  TABLE t1( c3, pk, c1 ) USING (
    PROCEDURE t1_delete FOR UPLOAD DELETE
  )
)

```

Defining stored procedures for updates

The stored procedure for updates must return a result set that includes two sets of values:

- The first set of values specifies the pre-image for the update (the values in the row the last time it was received from, or successfully uploaded to, the MobiLink server).
- The second set of values specifies the post-image of the update (the values the row should be updated to in the consolidated database).

This means that the stored procedure for updates must return a result set with twice as many columns as the insert or delete stored procedure.

Example

For a detailed explanation of how to define stored procedures for updates, see [“Scripted upload example” on page 323](#).

The following example creates a table called t1 and a publication called p1. The publication specifies WITH SCRIPTED UPLOAD and registers the stored procedure t1_update as the update procedure. The publication specifies three columns to be synchronized: pk, c1 and c3. The update procedure returns a result set with six columns. The first three columns contain the pre-image of the pk, c1 and c3 columns; the second three columns contain the post-image of the same columns. Note that in both cases the columns are ordered as they were when the table was created, not as they are ordered in the CREATE PUBLICATION statement.

```
CREATE TABLE t1(
  //Column ordering is taken from here
  pk integer primary key,
  c1 char( 30),
  c2 float,
  c3 double );

CREATE PROCEDURE t1_update (
RESULT( preimage_pk integer, preimage_c1 char(30), preimage_c3 double,
postimage_pk integer, postimage_c1 char(30), postimage_c3 double )
BEGIN
  ...
END

CREATE PUBLICATION WITH SCRIPTED UPLOAD p1 (
  // Order of columns here is ignored
  TABLE t1( c3, pk, c1 ) USING (
  PROCEDURE t1_update FOR UPLOAD UPDATE
  )
)
```

Creating publications for scripted upload

To create a scripted upload publication, use the keywords WITH SCRIPTED UPLOAD and specify the stored procedures in the USING clause.

If you do not define a stored procedure for a table in the scripted upload publication, no operations are uploaded for the table. You cannot use ALTER PUBLICATION to change a regular publication into a scripted upload publication.

Example

The following publication uses stored procedures to upload data for two tables, called t1 and t2. Inserts, deletes, and updates are uploaded for table t1. Only inserts are uploaded for table t2.

```
CREATE PUBLICATION pub WITH SCRIPTED UPLOAD (
  TABLE t1 (col1, col2, col3) USING (
    PROCEDURE my.t1_ui FOR UPLOAD INSERT,
    PROCEDURE my.t1_ud FOR UPLOAD DELETE,
    PROCEDURE my.t1_uu FOR UPLOAD UPDATE
  ),
  TABLE t2 USING (
    PROCEDURE my.t2_ui FOR UPLOAD INSERT
  )
)
```

See also

- “CREATE PUBLICATION statement [MobiLink] [SQL Remote]” [[SQL Anywhere Server - SQL Reference](#)]
- “ALTER PUBLICATION statement [MobiLink] [SQL Remote]” [[SQL Anywhere Server - SQL Reference](#)]

Scripted upload example

This example shows you how to set up a scripted upload that provides conflict detection. The example creates the consolidated and remote databases, stored procedures, publications and subscriptions that are required by scripted upload. 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 the consolidated database

Create a directory to hold the sample files. For example, call it *scriptedupload*. Open a command prompt and navigate to that directory.

(This example specifies file names and assume they are in the current directory. In a real application, you should specify the full path to the file.)

Run the following command to create a consolidated database:

```
dbinit consol.db
```

Next, run the following command to define an ODBC data source for the consolidated database:

```
dbdsn -w dsn_consol -y -c "uid=DBA;pwd=sql;dbf=consol.db;server=consol"
```

To use a database as a consolidated database, you must run a setup script that adds system tables, views, and stored procedures that are used by MobiLink. The following command sets up consol.db as a consolidated database:

```
dbisql -c "dsn=dsn_consol" %SQLANY12%\MobiLink\setup\synrsa.sql
```

Open Interactive SQL and connect to consol.db using the dsn_consol DSN. Run the following SQL statements. They create the employee table on the consolidated database, insert values into the table, and create the required synchronization scripts.

```
CREATE TABLE employee (
    id      unsigned integer primary key,
    name    varchar( 256),
    salary  numeric( 9, 2 )
);

INSERT INTO employee VALUES( 100, 'smith', 225000 );
COMMIT;

CALL ml_add_table_script( 'default', 'employee', 'upload_insert',
    'INSERT INTO employee ( id, name, salary ) VALUES ( {ml r.id}, {ml
r.name}, {ml r.salary} )' );

CALL ml_add_table_script( 'default', 'employee', 'upload_update',
    'UPDATE employee SET name = {ml r.name}, salary = {ml r.salary} WHERE
id = {ml r.id}' );

CALL ml_add_table_script( 'default', 'employee', 'upload_delete',
    'DELETE FROM employee WHERE id = {ml r.id}' );

CALL ml_add_table_script( 'default', 'employee', 'download_cursor',
    'SELECT * from employee' );
```

Create the remote database

At a command prompt in your samples directory, run the following command to create a remote database:

```
dbinit remote.db
```

Next, run the following command to define an ODBC data source:

```
dbdsn -w dsn_remote -y -c "uid=dba;pwd=sql;dbf=remote.db;server=remote"
```

In Interactive SQL, connect to remote.db using the dsn_remote DSN. Run the following set of statements to create objects in the remote database.

First, create the table to be synchronized. The insert_time and delete_time columns are not synchronized but contain information used by the upload stored procedures to determine which rows to upload.

```
CREATE TABLE employee (
    id      unsigned integer primary key,
    name    varchar( 256),
    salary  numeric( 9, 2 ),
    insert_time  timestamp default '1900-01-01'
);
```

Next, you need to define stored procedures and other things to handle the upload. You do this separately for inserts, deletes, and updates.

Handle inserts

First, create a trigger to set the insert_time on each row when it is inserted. This timestamp is used to determine if a row has been inserted since the last synchronization. This trigger is not fired when

dbmsync is applying downloaded inserts from the consolidated database because later in this example you set the FireTriggers extended option to off. Rows inserted by the download get an insert_time of 1900-01-01, the default value defined when the employee table was created. This value should always be before the start progress so those rows are not treated as new inserts and are not uploaded during the next synchronization.

```
CREATE TRIGGER emp_ins AFTER INSERT ON employee
REFERENCING NEW AS newrow
FOR EACH ROW
BEGIN
    UPDATE employee SET insert_time = CURRENT_TIMESTAMP
    WHERE id = newrow.id
END;
```

Next, create a procedure to return as a result set all the inserted rows to be uploaded. This procedure returns all rows that (based on the insert_time) have been inserted since the last successful upload but were not subsequently deleted. The time of the last successful upload is determined from the start progress value in the #hook_dict table. This example uses the default setting for the dbmsync extended option LockTables, which causes dbmsync to lock the tables being synchronized. As a result, you do not need to exclude rows inserted after the end progress: the table locks prevent any operations from occurring after the end progress, while the upload is built.

```
CREATE PROCEDURE employee_insert()
RESULT( id unsigned integer,
        name varchar( 256 ),
        salary numeric( 9,2 )
)
BEGIN
    DECLARE start_time timestamp;

    SELECT value
    INTO start_time
    FROM #hook_dict
    WHERE name = 'start progress as timestamp';

    // Upload as inserts all rows inserted after the start_time
    // that were not subsequently deleted
    SELECT id, name, salary
    FROM employee e
    WHERE insert_time > start_time AND
        NOT EXISTS( SELECT id FROM employee_delete ed WHERE ed.id = e.id );

END;
```

Handle updates

To handle uploads, you need to ensure that the correct pre-image is used based on the start progress when the upload was built.

First, create a table that maintains pre-images of updated rows. The pre-images are used when generating the scripted upload.

```
CREATE TABLE employee_preimages (
    id unsigned integer NOT NULL,
    name varchar( 256),
    salary numeric( 9, 2 ),
    img_time timestamp default CURRENT_TIMESTAMP,
    primary key( id, img_time )
);
```

Next, create a trigger to store a pre-image for each row when it is updated. As with the insert trigger, this trigger is not fired on download.

Note that this trigger stores a pre-image row each time a row is updated (unless two updates come so close together that they get the same timestamp). At first glance this looks wasteful. It would be tempting to only store a pre-image for the row if there is not already one in the table, and then count on the `sp_hook_dbmlsync_upload_end` hook to delete pre-images once they have been uploaded.

However, the `sp_hook_dbmlsync_upload_end` hook is not reliable for this purpose. The hook may not be called if a hardware or software failure stops dbmlsync after the upload is sent but before it is acknowledged, resulting in rows not being deleted from the pre-images table even though they have been successfully uploaded. Also, when a communication failure occurs dbmlsync may not receive an acknowledgement from the server for an upload. In this case, the upload status passed to the hook is 'unknown'. When this happens there is no way for the hook to tell if the pre-images table should be cleaned or left intact. By storing multiple pre-images, the correct one can always be selected based on the start progress when the upload is built.

```
CREATE TRIGGER emp_upd AFTER UPDATE OF name,salary ON employee
  REFERENCING OLD AS oldrow
  FOR EACH ROW
BEGIN
  INSERT INTO employee_preimages ON EXISTING SKIP VALUES(
    oldrow.id, oldrow.name, oldrow.salary, CURRENT_TIMESTAMP );
END;
```

Next, create an upload procedure to handle updates. This stored procedure returns one result set that has twice as many columns as the other scripts: it contains the pre-image (the values in the row the last time it was received from, or successfully uploaded to, the MobiLink server), and the post-image (the values to be entered into the consolidated database).

The pre-image is the earliest set of values in `employee_preimages` that was recorded after the `start_progress`. Note that this example does not correctly handle existing rows that are deleted and then reinserted. In a more complete solution, these would be uploaded as an update.

```
CREATE PROCEDURE employee_update()
RESULT(
  preimage_id unsigned integer,
  preimage_name varchar( 256),
  preimage_salary numeric( 9,2 ),
  postimage_id unsigned integer,
  postimage_name varchar( 256),
  postimage_salary numeric( 9,2 )
)
BEGIN
  DECLARE start_time timestamp;

  SELECT value
  INTO start_time
  FROM #hook_dict
  WHERE name = 'start progress as timestamp';

  // Upload as an update all rows that have been updated since
  // start_time that were not newly inserted or deleted.
  SELECT ep.id, ep.name, ep.salary, e.id, e.name, e.salary
  FROM employee e JOIN employee_preimages ep
  ON ( e.id = ep.id )
  // Do not select rows inserted since the start time. These should be
```



```

// uploaded as inserts.
WHERE insert_time <= start_time
// Do not upload deleted rows.
AND NOT EXISTS( SELECT id FROM employee_delete ed WHERE ed.id = e.id )
// Select the earliest pre-image after the start time.
AND ep.img_time = ( SELECT MIN( img_time )
                    FROM employee_preimages
                    WHERE id = ep.id
                    AND img_time > start_time );
END;

```

Handle deletes

First, create a table to maintain a list of deleted rows:

```

CREATE TABLE employee_delete (
  id          unsigned integer  primary key NOT NULL,
  name       varchar( 256 ),
  salary     numeric( 9, 2 ),
  delete_time timestamp
);

```

Next, create a trigger to populate the `employee_delete` table as rows are deleted from the `employee` table. This trigger is not called during download because later you set the `dbmsync` extended option `FireTriggers` to `false`. Note that this trigger assumes that a deleted row is never reinserted; therefore it does not deal with the same row being deleted more than once.

```

CREATE TRIGGER emp_del AFTER DELETE ON employee
REFERENCING OLD AS delrow
FOR EACH ROW
BEGIN
  INSERT INTO employee_delete
VALUES( delrow.id, delrow.name, delrow.salary, CURRENT_TIMESTAMP );
END;

```

The next SQL statement creates an upload procedure to handle deletes. This stored procedure returns a result set that contains the rows to delete on the consolidated database. The stored procedure uses the `employee_preimages` table so that if a row is updated and then deleted, the image uploaded for the delete is the last one that was successfully downloaded or uploaded.

```

CREATE PROCEDURE employee_delete()
RESULT( id unsigned integer,
        name varchar( 256 ),
        salary numeric( 9,2 )
      )
BEGIN
  DECLARE start_time timestamp;

  SELECT value
  INTO start_time
  FROM #hook_dict
  WHERE name = 'start progress as timestamp';

  // Upload as a delete all rows that were deleted after the
  // start_time that were not inserted after the start_time.
  // If a row was updated before it was deleted, then the row
  // to be deleted is the pre-image of the update.
  SELECT IF ep.id IS NULL THEN ed.id ELSE ep.id ENDIF,
         IF ep.id IS NULL THEN ed.name ELSE ep.name ENDIF,
         IF ep.id IS NULL THEN ed.salary ELSE ep.salary ENDIF
  FROM employee_delete ed LEFT OUTER JOIN employee_preimages ep

```

```

        ON( ed.id = ep.id AND ep.img_time > start_time )
WHERE
    // Only upload deletes that occurred since the last sync.
    ed.delete_time > start_time
    // Don't upload a delete for rows that were inserted since
    // the last upload and then deleted.
AND NOT EXISTS (
    SELECT id
      FROM employee e
     WHERE e.id = ep.id AND e.insert_time > start_time )
// Select the earliest preimage after the start time.
AND ( ep.id IS NULL OR ep.img_time = (SELECT MIN( img_time )
                                     FROM employee_preimages
                                     WHERE id = ep.id
                                     AND img_time > start_time ) );
END;

```

Clear out the pre-image table

Next, create an upload_end hook to clean up the employee_preimage and employee_delete tables when an upload is successful. This example uses the default setting for the dbmlsync extended option LockTables, so the tables are locked during synchronization. So, you do not have to worry about leaving rows in the tables for operations that occurred after the end_progress. Locking prevents such operations from occurring.

```

CREATE PROCEDURE sp_hook_dbmlsync_upload_end()
BEGIN
    DECLARE val    varchar(256);

    SELECT value
    INTO val
    FROM #hook_dict
    WHERE name = 'upload status';

    IF val = 'committed' THEN
        DELETE FROM employee_delete;
        DELETE FROM employee_preimages;
    END IF;
END;

```

Create a publication, MobiLink user, and subscription

The publication called pub1 uses the scripted upload syntax (WITH SCRIPTED UPLOAD). It creates an article for the employee table, and registers the three stored procedures you just created for use in the scripted upload. It creates a MobiLink user called u1, and a subscription between v1 and pub1. The extended option FireTriggers is set to off to prevent triggers from being fired on the remote database when the download is applied, which prevents downloaded changes from being uploaded during the next synchronization.

```

CREATE PUBLICATION pub1 WITH SCRIPTED UPLOAD (
TABLE employee( id, name, salary ) USING (
    PROCEDURE employee_insert FOR UPLOAD INSERT,
    PROCEDURE employee_update FOR UPLOAD UPDATE,
    PROCEDURE employee_delete FOR UPLOAD DELETE,
    )
)

CREATE SYNCHRONIZATION USER u1;

CREATE SYNCHRONIZATION SUBSCRIPTION TO pub1 FOR u1
TYPE 'tcpip'
ADDRESS 'host=localhost'

```

```
OPTION FireTriggers='off';  
SCRIPT VERSION 'default'
```

Demonstrate the scripted upload

Connect to the remote database and insert data to synchronize using scripted upload. For example, run the following SQL statements against the remote database in Interactive SQL:

```
INSERT INTO employee(id, name, salary) VALUES( 7, 'black', 700 );  
INSERT INTO employee(id, name, salary) VALUES( 8, 'anderson', 800 );  
INSERT INTO employee(id, name, salary) VALUES( 9, 'dilon', 900 );  
INSERT INTO employee(id, name, salary) VALUES( 10, 'dwit', 1000 );  
INSERT INTO employee(id, name, salary) VALUES( 11, 'dwit', 1100 );  
COMMIT;
```

At a command prompt, start the MobiLink server:

```
mlsrv12 -c "dsn=dsn_consol" -o mlserver.mls -v+ -dl -zu+
```

Start a synchronization using dbmlsync:

```
dbmlsync -c "dsn=dsn_remote" -k -uo -o remote.mlc -v+
```

You can now verify that the inserts were uploaded.

Example cleanup

To clean up your computer after completing the example, perform the following steps:

```
mlstop -h -w  
dbstop -y -c server=consol  
dbstop -y -c server=remote  
  
dberase -y consol.db  
dberase -y remote.db  
  
del remote.mlc mlserver.mls
```

Index

Symbols

- #hook_dict table
 - about MobiLink, 180
 - dbmlsync, 180
 - scripted upload, 318
- a option
 - MobiLink SQL Anywhere client utility (dbmlsync), 97
- ap option
 - MobiLink file transfer utility (mlfiletransfer), 21
 - MobiLink SQL Anywhere client utility (dbmlsync), 98
- ba option
 - MobiLink SQL Anywhere client utility (dbmlsync), 98
- bc option
 - MobiLink SQL Anywhere client utility (dbmlsync), 99
- be option
 - MobiLink SQL Anywhere client utility (dbmlsync), 99
- bg option
 - MobiLink SQL Anywhere client utility (dbmlsync), 99
- bk option
 - MobiLink SQL Anywhere client utility (dbmlsync), 100
- bkr option
 - MobiLink SQL Anywhere client utility (dbmlsync), 101
- c option
 - MobiLink SQL Anywhere client utility (dbmlsync), 101
- ci option
 - MobiLink SQL Anywhere client utility (dbmlsync), 102
- cl option
 - MobiLink SQL Anywhere client utility (dbmlsync), 102
- cm option
 - MobiLink SQL Anywhere client utility (dbmlsync), 103
- d option
 - MobiLink SQL Anywhere client utility (dbmlsync), 103
- dc option
 - MobiLink SQL Anywhere client utility (dbmlsync), 103
- dl option
 - MobiLink SQL Anywhere client utility (dbmlsync), 104
- do option
 - MobiLink SQL Anywhere client utility (dbmlsync), 104
- drs option
 - MobiLink SQL Anywhere client utility (dbmlsync), 105
- ds option
 - MobiLink SQL Anywhere client utility (dbmlsync), 106
- e adr
 - dbmlsync extended option, 128
 - options for, 23
- e cd
 - dbmlsync extended option, 130
- e CommunicationAddress
 - dbmlsync extended option, 128
 - options for, 23
- e CommunicationType
 - dbmlsync extended option, 129
- e ConflictRetries
 - dbmlsync extended option, 129
- e ContinueDownload
 - dbmlsync extended option, 130
- e cr
 - dbmlsync extended option, 129
- e ctp
 - dbmlsync extended option, 129
- e dir
 - dbmlsync extended option, 142
- e DisablePolling
 - dbmlsync extended option, 131
- e DownloadOnly
 - dbmlsync extended option, 132
- e DownloadReadSize
 - dbmlsync extended option, 133
- e drs
 - dbmlsync extended option, 133
- e ds
 - dbmlsync extended option, 132
- e eh

- dbmsync extended option, 136
- e el
 - dbmsync extended option, 134
- e ErrorLogSendLimit
 - dbmsync extended option, 134
- e FireTriggers
 - dbmsync extended option, 135
- e ft
 - dbmsync extended option, 135
- e HoverRescanThreshold
 - dbmsync extended option, 135
- e hrt
 - dbmsync extended option, 135
- e IgnoreHookErrors
 - dbmsync extended option, 136
- e IgnoreScheduling
 - dbmsync extended option, 136
- e inc
 - dbmsync extended option, 137
- e Increment
 - dbmsync extended option, 137
- e isc
 - dbmsync extended option, 136
- e LockTables
 - dbmsync extended option, 138
- e lt
 - dbmsync extended option, 138
- e MirrorLogDirectory
 - dbmsync extended option, 139
- e mld
 - dbmsync extended option, 139
- e mn
 - dbmsync extended option, 140
- e MobiLinkPwd
 - dbmsync extended option, 140
- e mp
 - dbmsync extended option, 140
- e NewMobiLinkPwd
 - dbmsync extended option, 140
- e NoSyncOnStartup
 - dbmsync extended option, 141
- e nss
 - dbmsync extended option, 141
- e OfflineDirectory
 - dbmsync extended option, 142
- e option
 - MobiLink SQL Anywhere client utility (dbmsync), 106
- e p
 - dbmsync extended option, 131
- e PollingPeriod
 - dbmsync extended option, 142
- e pp
 - dbmsync extended option, 142
- e sa
 - dbmsync extended option, 146
- e sch
 - dbmsync extended option, 143
- e Schedule
 - dbmsync extended option, 143
- e scn
 - dbmsync extended option, 146
- e ScriptVersion
 - dbmsync extended option, 145
- e SendColumnNames
 - dbmsync extended option, 146
- e SendDownloadAck
 - dbmsync extended option, 146
- e SendTriggers
 - dbmsync extended option, 147
- e st
 - dbmsync extended option, 147
- e sv
 - dbmsync extended option, 145
- e TableOrder
 - dbmsync extended option, 148
- e TableOrderChecking
 - dbmsync extended option, 149
- e toc
 - dbmsync extended option, 149
- e tor
 - dbmsync extended option, 148
- e uo
 - dbmsync extended option, 150
- e UploadOnly
 - dbmsync extended option, 150
- e v
 - dbmsync extended option, 150
- e Verbose
 - dbmsync extended option, 150
- e VerboseHooks
 - dbmsync extended option, 151
- e VerboseMin
 - dbmsync extended option, 152
- e VerboseOptions
 - dbmsync extended option, 153

-
- e VerboseRowCounts
 - dbmlsync extended option, 154
 - e VerboseRowValues
 - dbmlsync extended option, 155
 - e VerboseUpload
 - dbmlsync extended option, 156
 - e vm
 - dbmlsync extended option, 152
 - e vn
 - dbmlsync extended option, 154
 - e vo
 - dbmlsync extended option, 153
 - e vr
 - dbmlsync extended option, 155
 - e vs
 - dbmlsync extended option, 151
 - e vu
 - dbmlsync extended option, 156
 - eh option
 - MobiLink SQL Anywhere client utility (dbmlsync), 107
 - ek option
 - MobiLink SQL Anywhere client utility (dbmlsync), 107
 - ep option
 - MobiLink SQL Anywhere client utility (dbmlsync), 108
 - eu option
 - MobiLink SQL Anywhere client utility (dbmlsync), 108
 - f option
 - MobiLink file transfer utility (mlfiletransfer), 21
 - g option
 - MobiLink file transfer utility (mlfiletransfer), 21
 - is option
 - MobiLink SQL Anywhere client utility (dbmlsync), 108
 - k option
 - MobiLink Microsoft ActiveSync provider utility (mlasinst), 18
 - MobiLink SQL Anywhere client utility (dbmlsync) (deprecated), 109
 - l option
 - MobiLink SQL Anywhere client utility (dbmlsync), 109
 - lf option
 - MobiLink file transfer utility (mlfiletransfer), 21
 - lp option
 - MobiLink file transfer utility (mlfiletransfer), 21
 - mn option
 - MobiLink SQL Anywhere client utility (dbmlsync), 109
 - mp option
 - MobiLink SQL Anywhere client utility (dbmlsync), 110
 - n option
 - MobiLink Microsoft ActiveSync provider utility (mlasinst), 18
 - MobiLink SQL Anywhere client utility (dbmlsync), 110
 - o option
 - MobiLink SQL Anywhere client utility (dbmlsync), 111
 - os option
 - MobiLink SQL Anywhere client utility (dbmlsync), 111
 - ot option
 - MobiLink SQL Anywhere client utility (dbmlsync), 112
 - p option
 - MobiLink file transfer utility (mlfiletransfer), 21
 - MobiLink SQL Anywhere client utility (dbmlsync), 112
 - pc option
 - MobiLink SQL Anywhere client utility (dbmlsync), 113
 - pd option
 - MobiLink SQL Anywhere client utility (dbmlsync), 114
 - pi option
 - MobiLink SQL Anywhere client utility (dbmlsync), 114
 - po option
 - MobiLink SQL Anywhere client utility (dbmlsync), 115
 - pp option
 - MobiLink SQL Anywhere client utility (dbmlsync), 115
 - q option
 - MobiLink SQL Anywhere client utility (dbmlsync), 116
 - qc option
 - MobiLink SQL Anywhere client utility (dbmlsync), 116
 - r option
 - MobiLink file transfer utility (mlfiletransfer), 21

- MobiLink SQL Anywhere client utility (dbmlsync), 116
 - ra option
 - MobiLink SQL Anywhere client utility (dbmlsync), 116
 - rb option
 - MobiLink SQL Anywhere client utility (dbmlsync), 116
 - s option
 - MobiLink file transfer utility (mlfiletransfer), 21
 - MobiLink SQL Anywhere client utility (dbmlsync), 117
 - sc option
 - MobiLink SQL Anywhere client utility (dbmlsync), 118
 - sm option
 - MobiLink SQL Anywhere client utility (dbmlsync), 118
 - sp option
 - MobiLink SQL Anywhere client utility (dbmlsync), 119
 - tu option
 - MobiLink SQL Anywhere client utility (dbmlsync), 119
 - u option
 - MobiLink file transfer utility (mlfiletransfer), 21
 - MobiLink Microsoft ActiveSync provider utility (mlasinst), 18
 - MobiLink SQL Anywhere client utility (dbmlsync), 120
 - ud option
 - MobiLink SQL Anywhere client utility (dbmlsync), 121
 - ui option
 - MobiLink SQL Anywhere client utility (dbmlsync), 122
 - uo option
 - MobiLink SQL Anywhere client utility (dbmlsync), 122
 - urc option
 - MobiLink SQL Anywhere client utility (dbmlsync), 123
 - ux option
 - MobiLink SQL Anywhere client utility (dbmlsync), 123
 - v option
 - MobiLink file transfer utility (mlfiletransfer), 21
 - MobiLink Microsoft ActiveSync provider utility (mlasinst), 18
 - MobiLink SQL Anywhere client utility (dbmlsync), 123
 - v+ option
 - MobiLink SQL Anywhere client utility (dbmlsync), 123
 - vc option
 - MobiLink SQL Anywhere client utility (dbmlsync), 123
 - vn option
 - MobiLink SQL Anywhere client utility (dbmlsync), 123
 - vo option
 - MobiLink SQL Anywhere client utility (dbmlsync), 123
 - vp option
 - MobiLink SQL Anywhere client utility (dbmlsync), 123
 - vr option
 - MobiLink SQL Anywhere client utility (dbmlsync), 123
 - vs option
 - MobiLink SQL Anywhere client utility (dbmlsync), 123
 - vu option
 - MobiLink SQL Anywhere client utility (dbmlsync), 123
 - wc option
 - MobiLink SQL Anywhere client utility (dbmlsync), 124
 - x option
 - MobiLink file transfer utility (mlfiletransfer), 21
 - MobiLink SQL Anywhere client utility (dbmlsync), 125
 - .NET
 - MobiLink user authentication, 15
 - @data option
 - MobiLink SQL Anywhere client utility (dbmlsync), 97
- A**
- a_dbtools_info structure
 - initializing, 305
 - a_sync_db structure
 - initializing, 305
 - introduction, 305

- a_syncpub structure
 - introduction, 305
- ActiveX
 - MobiLink dbmlsync integration component, 276
- add user wizard
 - MobiLink plug-in, 6
- adding
 - articles for MobiLink SQL Anywhere clients, 72
 - columns to remote MobiLink databases, 60
 - MobiLink users to a SQL Anywhere client, 74
 - MobiLink users to the consolidated database, 5
 - tables to remote MobiLink SQL Anywhere databases, 60
- adr dbmlsync extended option
 - about, 128
 - options for, 23
- altering
 - articles for MobiLink SQL Anywhere clients, 72
 - MobiLink publications for SQL Anywhere clients, 72
 - subscriptions for SQL Anywhere clients, 78
- altering existing publications
 - MobiLink SQL Anywhere clients, 72
- altering MobiLink subscriptions
 - SQL Anywhere clients, 78
- ApplyDnldFile
 - MobiLink synchronization profile option, 158
- args option
 - MobiLink Microsoft ActiveSync provider utility (mlasinst), 18
- articles
 - adding for MobiLink SQL Anywhere clients, 72
 - altering for MobiLink SQL Anywhere clients, 72
 - creating for MobiLink SQL Anywhere clients, 67
 - MobiLink synchronization subscriptions, 77
 - removing from MobiLink SQL Anywhere clients, 72
- authenticate_user
 - about, 14
 - authentication process, 12
 - using predefined scripts, 15
- authenticating
 - MobiLink to external servers, 15
- authenticating MobiLink users
 - about, 4
- authentication
 - MobiLink authentication process, 12
 - MobiLink users, 4
- authentication process
 - MobiLink, 12
- AuthParms
 - MobiLink synchronization profile option, 158
- auto-dial
 - MobiLink client connection option, 45
- AUTOINCREMENT
 - (*see also* GLOBAL AUTOINCREMENT)

B

- Background
 - MobiLink synchronization profile option, 158
- BackgroundRetry
 - MobiLink synchronization profile option, 158
- backups
 - restoring remote databases, 116
- BeginDownload event
 - dbmlsync integration component, 285
- BeginLogScan event
 - dbmlsync integration component, 286
- BeginSynchronization event
 - dbmlsync integration component, 286
- BeginUpload event
 - dbmlsync integration component, 287
- buffer size
 - MobiLink client connection option, 28
- buffer_size protocol option
 - MobiLink client connection option, 28
- bugs
 - providing feedback, viii

C

- cac authentication
 - MobiLink client connection option, 43
- cac option
 - MobiLink clients, 43
- CacheInit
 - MobiLink synchronization profile option, 158
- CacheMax
 - MobiLink synchronization profile option, 158
- CacheMin
 - MobiLink synchronization profile option, 158
- CancelSync method
 - DbmlsyncClient class [Dbmlsync .NET API], 255
 - DbmlsyncClient class [Dbmlsync C++ API], 234
- cd dbmlsync extended option
 - about, 130

- certificate fields
 - MobiLink TLS certificate_company option, 29
 - MobiLink TLS certificate_name option, 30
 - MobiLink TLS certificate_unit option, 32
- certificate option
 - MobiLink client connection option, 43
- certificate_company protocol option
 - MobiLink client connection option, 29
- certificate_name protocol option
 - MobiLink client connection option, 30
- certificate_unit protocol option
 - MobiLink client connection option, 32
- changing passwords
 - MobiLink, 8
- checking table order
 - dbmsync extended option, 149
- class names
 - Microsoft ActiveSync, 124
- class option
 - MobiLink Microsoft ActiveSync provider utility (mlasinst), 18
- client databases
 - MobiLink dbmsync options, 93
- client event-hook procedures
 - MobiLink SQL Anywhere clients, 178
- client network protocol options
 - MobiLink, 23
- client_port protocol option
 - MobiLink client connection option, 32
- clients
 - MobiLink SQL Anywhere client (dbmsync), 93
 - SQL Anywhere as MobiLink, 1
 - SQL Anywhere MobiLink clients, 63
 - UltraLite applications as MobiLink, 1
- column-wise partitioning
 - MobiLink SQL Anywhere clients, 69
- ColumnCount property
 - dbmsync integration component, 304
- ColumnName
 - dbmsync integration component, 302
- columns
 - adding to remote MobiLink databases, 60
- ColumnValue property
 - dbmsync integration component, 303
- command line
 - starting dbmsync, 93
- command line utilities
 - MobiLink clients, 18
 - MobiLink file transfer (mfiletransfer) syntax, 21
 - MobiLink Microsoft ActiveSync provider utility (mlasinst) syntax, 18
 - MobiLink SQL Anywhere client utility (dbmsync) syntax, 93
- command prompts
 - conventions, vii
 - curly braces, vii
 - environment variables, vii
 - parentheses, vii
 - quotes, vii
 - semicolons, vii
- command shells
 - conventions, vii
 - curly braces, vii
 - environment variables, vii
 - parentheses, vii
 - quotes, vii
- COMMIT statement
 - event-hook procedures, 180
- CommunicationAddress dbmsync extended option
 - about, 128
 - options for, 23
- communications
 - MobiLink clients, 23
 - MobiLink dbmsync -c option, 101
 - MobiLink dbmsync adr option, 128
 - MobiLink dbmsync ctp option, 129
 - specifying for MobiLink, 2
- CommunicationType dbmsync extended option
 - about, 129
- compression
 - MobiLink client connection option, 33
- compression protocol option
 - MobiLink client connection option, 33
 - UltraLite deployment requirements for, 28
- concurrency
 - MobiLink SQL Anywhere clients, 82
- configuring
 - MobiLink user properties for SQL Anywhere clients, 75
 - SQL Anywhere remote databases for Microsoft ActiveSync, 84
- ConflictRetries dbmsync extended option
 - about, 129
 - concurrency during synchronization, 82
- Connect method
 - DbmsyncClient class [Dbmsync .NET API], 257

- DbmlsyncClient class [Dbmlsync C++ API], 235
- connecting
 - MobiLink clients, 23
 - MobiLink dbmlsync -c option, 101
 - MobiLink dbmlsync adr option, 128
 - MobiLink dbmlsync ctp option, 129
- connection failures
 - MobiLink dbmlsync clients, 182
- connection options
 - dbmlsync, 128
- connection parameters
 - MobiLink clients, 23
 - MobiLink SQL Anywhere clients, 81
 - priority order for MobiLink SQL Anywhere clients, 126
- connection strings
 - MobiLink dbmlsync, 101
- connections
 - MobiLink clients, 23
 - MobiLink dbmlsync -c option, 101
 - MobiLink dbmlsync adr option, 128
 - MobiLink dbmlsync ctp option, 129
- connections for event-hook procedures
 - SQL Anywhere clients, 182
- ConnectMobilink event
 - dbmlsync integration component, 288
- consistency
 - (*see also* synchronization)
- ContinueDownload
 - MobiLink synchronization profile option, 158
- ContinueDownload dbmlsync extended option
 - about, 130
- conventions
 - command prompts, vii
 - command shells, vii
 - documentation, v
 - file names in documentation, vi
 - operating systems, v
 - Unix, v
 - Windows, v
 - Windows CE, v
 - Windows Mobile, v
- cr dbmlsync extended option
 - about, 129
- create article wizard
 - using in MobiLink, 73
- create MobiLink user wizard
 - using, 74
- CREATE PUBLICATION statement
 - SQL Anywhere database usage, 67
- create publication wizard
 - column-wise partitioning in MobiLink, 69
 - row-wise partitioning in MobiLink SQL Anywhere clients, 70
- CREATE SYNCHRONIZATION SUBSCRIPTION statement
 - Microsoft ActiveSync for MobiLink SQL Anywhere clients, 84
- CREATE SYNCHRONIZATION USER statement
 - Microsoft ActiveSync for MobiLink SQL Anywhere clients, 84
- create user wizard
 - MobiLink plug-in, 6
- CreateDnldFile
 - MobiLink synchronization profile option, 158
- creating
 - articles for MobiLink SQL Anywhere clients, 67
 - MobiLink users, 5
 - MobiLink users in SQL Anywhere clients, 74
 - publications for MobiLink SQL Anywhere clients, 67
 - publications with column-wise partitioning for MobiLink SQL Anywhere clients, 69
 - publications with row-wise partitioning for MobiLink SQL Anywhere clients, 70
 - publications with whole tables for MobiLink SQL Anywhere clients, 68
 - SQL Anywhere remote databases, 63
- creating and registering MobiLink users
 - about, 5
- creating MobiLink users
 - about, 5
 - about SQL Anywhere clients, 74
- creating MobiLink users in the remote database
 - about, 74
- creating publications for scripted upload
 - about, 322
- creating remote databases
 - SQL Anywhere clients, 63
- creating synchronization subscriptions
 - SQL Anywhere clients, 77
- ctp dbmlsync extended option
 - about, 129
- custom authentication
 - MobiLink clients, 14
- custom headers

- MobiLink client connection option, 34
- custom user authentication
 - MobiLink clients, 14
- custom_header protocol option
 - MobiLink client connection option, 34
- customizing
 - SQL Anywhere client synchronization process, 178
- customizing dbmlsync synchronization
 - MobiLink SQL Anywhere clients, 89
- customizing the client synchronization process
 - SQL Anywhere clients, 178

D

- data consistency
 - (*see also* synchronization)
- database tools interface
 - (*see also* DBTools interface)
 - dbmlsync, 304
 - setting up for dbmlsync, 305
- database tools interface for dbmlsync
 - about, 304
- databases
 - MobiLink remote databases, 1
- dbmlsync (*see* dbmlsync utility)
 - options, 93
- Dbmlsync .NET API
 - DbmlsyncClient class, 254
 - DBSC_CancelRet enumeration, 268
 - DBSC_ErrorInfo structure, 274
 - DBSC_ErrorType enumeration, 268
 - DBSC_Event structure, 275
 - DBSC_EventType enumeration, 271
 - DBSC_GetEventRet enumeration, 272
 - DBSC_ShutdownType enumeration, 273
 - DBSC_StartType enumeration, 274
- Dbmlsync .NET API reference
 - iAnywhere.MobiLink.Client namespace, 253
- Dbmlsync API
 - architecture, 90
 - interfaces, 90
 - introduction, 90
- Dbmlsync C++ API
 - DbmlsyncClient class, 232
 - DBSC_CancelRet enumeration, 245
 - DBSC_ErrorInfo structure, 251
 - DBSC_ErrorType enumeration, 246
 - DBSC_Event structure, 252
 - DBSC_EventType enumeration, 248
 - DBSC_GetEventRet enumeration, 249
 - DBSC_ShutdownType enumeration, 250
 - DBSC_StartType enumeration, 250
- Dbmlsync C++ API reference
 - dbmlsynccli.h header file, 231
- dbmlsync client event hooks
 - introducing, 89
- dbmlsync error
 - handling, 182
- dbmlsync extended options
 - about, 126
 - using, 80
- dbmlsync integration component
 - about, 276
 - events, 285
 - IRowTransfer interface, 300
 - setup, 277
 - supported platforms, 277
- dbmlsync integration component methods
 - about, 278
- dbmlsync integration component properties
 - about, 279
- dbmlsync message log
 - about, 90
- dbmlsync network protocol options
 - about, 81
- dbmlsync utility
 - #hook_dict table, 180
 - alphabetical list of options, 93
 - changing passwords, 8
 - concurrency, 82
 - connecting to the MobiLink server, 128
 - connecting to the remote database, 101
 - customizing MobiLink synchronization, 178
 - DBTools interface, 304
 - error handling event hooks, 182
 - event hooks, 178
 - extended options, 126
 - hooks, 178
 - initiating synchronization from an application, 83
 - integration component, 276
 - Mac OS X, 91
 - Microsoft ActiveSync for MobiLink SQL Anywhere clients, 83
 - options, 93
 - passwords, 7
 - permissions, 80

programming interface, 90
 progress offsets, 66
 sp_hook_dbmlsync_abort hook, 183
 sp_hook_dbmlsync_all_error, 185
 sp_hook_dbmlsync_begin, 188
 sp_hook_dbmlsync_communication_error, 189
 sp_hook_dbmlsync_delay, 191
 sp_hook_dbmlsync_download_begin, 194
 sp_hook_dbmlsync_download_end, 195
 sp_hook_dbmlsync_download_log_ri_violation, 196
 sp_hook_dbmlsync_download_ri_violation, 198
 sp_hook_dbmlsync_download_table_begin, 200
 sp_hook_dbmlsync_download_table_end, 201
 sp_hook_dbmlsync_end, 203
 sp_hook_dbmlsync_log_rescan, 205
 sp_hook_dbmlsync_logscan_begin, 207
 sp_hook_dbmlsync_logscan_end, 208
 sp_hook_dbmlsync_misc_error, 210
 sp_hook_dbmlsync_ml_connect_failed, 213
 sp_hook_dbmlsync_process_exit_code, 216
 sp_hook_dbmlsync_schema_upgrade, 217
 sp_hook_dbmlsync_set_extended_options, 219
 sp_hook_dbmlsync_set_ml_connect_info, 220
 sp_hook_dbmlsync_set_upload_end_progress, 222
 sp_hook_dbmlsync_sql_error, 224
 sp_hook_dbmlsync_upload_begin, 226
 sp_hook_dbmlsync_upload_end, 227
 sp_hook_dbmlsync_validate_download_file, 229
 syntax, 93
 transaction logs, 81
 using, 79

dbmlsynccli.h header file
 Dbmlsync C++ API reference, 231

DbmlsyncClient class [Dbmlsync .NET API]
 CancelSync method, 255
 Connect method, 257
 description, 254
 Disconnect method, 258
 Fini method, 258
 GetErrorInfo method, 259
 GetEvent method, 259
 GetProperty method, 260
 Init method, 261
 InstantiateClient method, 262
 Ping method, 262
 SetProperty method, 262
 ShutdownServer method, 264
 StartServer method, 265
 Sync method, 266
 WaitForServerShutdown method, 267

DbmlsyncClient class [Dbmlsync C++ API]
 CancelSync method, 234
 Connect method, 235
 description, 232
 Disconnect method, 236
 Fini method, 237
 FreeEventInfo method, 237
 GetErrorInfo method, 238
 GetEvent method, 238
 GetProperty method, 239
 Init method, 239
 InstantiateClient method, 240
 Ping method, 240
 SetProperty method, 241
 ShutdownServer method, 242
 StartServer method, 243
 Sync method, 244
 WaitForServerShutdown method, 245

dbmlsynccom.dll
 dbmlsync integration component, 276

dbmlsynccomg.dll
 dbmlsync integration component, 276

DBSC_CancelRet enumeration [Dbmlsync .NET API]
 description, 268

DBSC_CancelRet enumeration [Dbmlsync C++ API]
 description, 245

DBSC_ErrorInfo structure [Dbmlsync .NET API]
 description, 274

DBSC_ErrorInfo structure [Dbmlsync C++ API]
 description, 251

DBSC_ErrorType enumeration [Dbmlsync .NET API]
 description, 268

DBSC_ErrorType enumeration [Dbmlsync C++ API]
 description, 246

DBSC_Event structure [Dbmlsync .NET API]
 description, 275

DBSC_Event structure [Dbmlsync C++ API]
 description, 252

DBSC_EventType enumeration [Dbmlsync .NET API]
 description, 271

DBSC_EventType enumeration [Dbmlsync C++ API]
 description, 248

DBSC_GetEventRet enumeration [Dbmlsync .NET API]
 description, 272

- DBSC_GetEventRet enumeration [Dbmsync C++ API]
 - description, 249
- DBSC_ShutdownType enumeration [Dbmsync .NET API]
 - description, 273
- DBSC_ShutdownType enumeration [Dbmsync C++ API]
 - description, 250
- DBSC_StartType enumeration [Dbmsync .NET API]
 - description, 274
- DBSC_StartType enumeration [Dbmsync C++ API]
 - description, 250
- DBSynchronizeLog function
 - introduction, 305
- DBTools interface
 - (*see also* database tools interface)
 - dbmsync, 304
 - setting up for dbmsync, 305
 - synchronizing SQL Anywhere clients, 83
- DBTools interface for dbmsync
 - about, 304
- dbtools.h
 - synchronizing SQL Anywhere clients, 83
- DBToolsFini function
 - using, 308
- DBToolsInit function
 - starting dbtools, 305
- DCX
 - about, v
- DDL
 - remote MobiLink databases, 58
- debugging
 - MobiLink dbmsync log, 90
- default_internet
 - network_name protocol option setting, 46
- default_work
 - network_name protocol option setting, 46
- deleting
 - articles from MobiLink SQL Anywhere clients, 72
 - MobiLink users from SQL Anywhere clients, 76
 - publications from MobiLink SQL Anywhere clients, 74
- deploying
 - MobiLink SQL Anywhere clients, 63
 - troubleshooting MobiLink deployment of SQL Anywhere clients, 66
- deploying remote databases
 - MobiLink SQL Anywhere clients, 63
- DetailedInfoMessageEnabled property
 - dbmsync integration component, 282
- developer centers
 - finding out more and requesting technical support, ix
- developer community
 - newsgroups, viii
- dial-up
 - dbmsync connection, 128
 - MobiLink client protocol options, 23
- dir dbmsync extended option
 - about, 142
- DisablePolling dbmsync extended option
 - about, 131
- Disconnect method
 - DbmsyncClient class [Dbmsync .NET API], 258
 - DbmsyncClient class [Dbmsync C++ API], 236
- DisconnectMobilink event
 - dbmsync integration component, 288
- DispatchChannelSize property
 - dbmsync integration component, 285
- dllapi.h
 - DBTools interface for dbmsync, 308
- DnldFileExtra
 - MobiLink synchronization profile option, 158
- DocCommentXchange (DCX)
 - about, v
- documentation
 - conventions, v
 - SQL Anywhere, v
- DoneExecution event
 - dbmsync integration component, 289
- download continues
 - dbmsync -dc option, 103
- download only
 - (*see also* download-only)
- download only synchronization
 - (*see also* download-only synchronization)
- download-only
 - dbmsync -ds option, 106
 - dbmsync DownloadOnly extended option, 132
 - differences between approaches, 71
 - publications, 71
- download-only publications
 - about, 71
- download-only synchronization
 - dbmsync -ds option, 106

- dbmsync DownloadOnly extended option, 132
- DownloadEventsEnabled property
 - dbmsync integration component, 280
- downloading rows
 - resolving MobiLink RI violations, 196
- DownloadOnly
 - MobiLink synchronization profile option, 158
- DownloadOnly dbmsync extended option
 - about, 132
- DownloadReadSize
 - MobiLink synchronization profile option, 158
- DownloadReadSize dbmsync extended option
 - about, 133
- DownloadRow event
 - dbmsync integration component, 290
- DROP PUBLICATION statement
 - MobiLink using, 74
- DROP SYNCHRONIZATION SUBSCRIPTION statement
 - using, 78
- dropping
 - MobiLink subscriptions from SQL Anywhere clients, 78
 - MobiLink users from SQL Anywhere clients, 76
- dropping MobiLink subscriptions
 - SQL Anywhere clients, 78
- dropping MobiLink users
 - SQL Anywhere clients, 76
- dropping publications
 - MobiLink SQL Anywhere clients, 74
- drs dbmsync extended option
 - about, 133
- ds dbmsync extended option
 - about, 132
- dst option
 - MobiLink Microsoft ActiveSync provider utility (mlasinst), 18

E

- e2ee_public_key protocol option
 - MobiLink client connection option, 36
- e2ee_type protocol option
 - MobiLink client connection option, 35
- ECC
 - MobiLink clients, 51
- ECC protocol option
 - MobiLink clients, 51
- eh dbmsync extended option
 - about, 136
- el dbmsync extended option
 - about, 134
- end-to-end encryption public key
 - MobiLink client connection option, 36
- end-to-end encryption type
 - MobiLink client connection option, 35
- EndDownload event
 - dbmsync integration component, 291
- EndLogScan event
 - dbmsync integration component, 291
- EndSynchronization event
 - dbmsync integration component, 292
- EndUpload event
 - dbmsync integration component, 293
- environment variables
 - command prompts, vii
 - command shells, vii
- ErrorLogSendLimit dbmsync extended option
 - about, 134
- ErrorMessageEnabled property
 - dbmsync integration component, 281
- errors
 - MobiLink dbmsync clients, 182
- event arguments
 - SQL Anywhere clients, 180
- event hook sequence
 - SQL Anywhere clients, 178
- event hooks
 - #hook_dict table, 180
 - about, 178
 - commits not allowed, 180
 - connections, 182
 - customizing the SQL Anywhere client synchronization process, 178
 - error handling, 182
 - event arguments, 180
 - event hook sequence, 178
 - fatal errors, 182
 - ignoring errors, 183
 - procedure owner, 180
 - rollbacks not allowed, 180
 - sp_hook_dbmsync_abort, 183
 - sp_hook_dbmsync_all_error, 185
 - sp_hook_dbmsync_begin, 188
 - sp_hook_dbmsync_communication_error, 189
 - sp_hook_dbmsync_delay, 191

- sp_hook_dbmlsync_download_begin, 194
- sp_hook_dbmlsync_download_log_ri_violation, 196
- sp_hook_dbmlsync_download_ri_violation, 198
- sp_hook_dbmlsync_download_table_begin, 200
- sp_hook_dbmlsync_download_table_end, 201
- sp_hook_dbmlsync_end, 203
- sp_hook_dbmlsync_log_rescan, 205
- sp_hook_dbmlsync_logscan_begin, 207
- sp_hook_dbmlsync_logscan_end, 208
- sp_hook_dbmlsync_misc_error, 210
- sp_hook_dbmlsync_ml_connect_failed, 213
- sp_hook_dbmlsync_process_exit_code, 216
- sp_hook_dbmlsync_schema_upgrade, 217
- sp_hook_dbmlsync_set_extended_options, 219
- sp_hook_dbmlsync_set_ml_connect_info, 220
- sp_hook_dbmlsync_set_upload_end_progress, 222
- sp_hook_dbmlsync_sql_error, 224
- sp_hook_dbmlsync_upload_begin, 226
- sp_hook_dbmlsync_upload_end, 227
- sp_hook_dbmlsync_validate_download_file, 229
 - using, 180
- event hooks for SQL Anywhere clients
 - about, 178
- event-hook procedure owner
 - SQL Anywhere clients, 180
- event-hooks
 - sp_hook_dbmlsync_begin, 194
 - sp_hook_dbmlsync_download_end, 195
- EventChannelSize property
 - dbmlsync integration component, 284
- events
 - dbmlsync integration component, 285
- examples
 - MobiLink scripted upload, 323
- exit codes
 - dbmlsync [sp_hook_dbmlsync_abort], 183
 - dbmlsync
 - [sp_hook_dbmlsync_process_exit_code], 216
- ExitCode property
 - dbmlsync integration component, 284
- extended options
 - configuring at SQL Anywhere clients, 75
 - dbmlsync, 126
 - priority order for SQL Anywhere clients, 126
- external authenticator properties
 - MobiLink, 17
- external servers
 - authenticating to in MobiLink applications, 15
- ExtOpt
 - MobiLink synchronization profile option, 158
- F**
- failover
 - MobiLink SQL Anywhere clients using
 - sp_hook_dbmlsync_ml_connect_failed, 213
- feedback
 - documentation, viii
 - providing, viii
 - reporting an error, viii
 - requesting an update, viii
- file transfers
 - MobiLink file transfer utility (mlfiletransfer), 21
- file-based downloads
 - dbmlsync -bc option, 99
 - dbmlsync -be option, 99
 - dbmlsync -bg option, 100
- finding out more and requesting technical assistance
 - technical support, viii
- Fini method
 - DbmlsyncClient class [Dbmlsync .NET API], 258
 - DbmlsyncClient class [Dbmlsync C++ API], 237
- FIPS
 - MobiLink client connection option, 36
- FIPS protocol option
 - MobiLink client connection option, 36
 - MobiLink clients, 51
- FireTriggers dbmlsync extended option
 - about, 135
- first synchronization always works
 - dbmlsync, 66
- FreeEventInfo method
 - DbmlsyncClient class [Dbmlsync C++ API], 237
- ft dbmlsync extended option
 - about, 135
- G**
- GetErrorInfo method
 - DbmlsyncClient class [Dbmlsync .NET API], 259
 - DbmlsyncClient class [Dbmlsync C++ API], 238
- GetEvent method
 - DbmlsyncClient class [Dbmlsync .NET API], 259
 - DbmlsyncClient class [Dbmlsync C++ API], 238
- GetProperty method
 - DbmlsyncClient class [Dbmlsync .NET API], 260

DbmlsyncClient class [Dbmlsync C++ API], 239
getting help
 technical support, viii
getting started
 SyncConsole, 91
GLOBAL AUTOINCREMENT
 (see also AUTOINCREMENT)

H

handling errors
 MobiLink dbmlsync clients, 182
handling errors and warnings in event hook procedures
 MobiLink dbmlsync clients, 182
help
 technical support, viii
hooks
 about dbmlsync event hooks, 178
 error handling, 182
 ignoring errors, 136
 sp_hook_dbmlsync_abort, 183
 sp_hook_dbmlsync_all_error, 185
 sp_hook_dbmlsync_begin, 188
 sp_hook_dbmlsync_communication_error, 189
 sp_hook_dbmlsync_delay, 191
 sp_hook_dbmlsync_download_begin, 194
 sp_hook_dbmlsync_download_end, 195
 sp_hook_dbmlsync_download_log_ri_violation, 196
 sp_hook_dbmlsync_download_ri_violation, 198
 sp_hook_dbmlsync_download_table_begin, 200
 sp_hook_dbmlsync_download_table_end, 201
 sp_hook_dbmlsync_end, 203
 sp_hook_dbmlsync_log_rescan, 205
 sp_hook_dbmlsync_logscan_begin, 207
 sp_hook_dbmlsync_logscan_end, 208
 sp_hook_dbmlsync_misc_error, 210
 sp_hook_dbmlsync_ml_connect_failed, 213
 sp_hook_dbmlsync_process_exit_code, 216
 sp_hook_dbmlsync_schema_upgrade, 217
 sp_hook_dbmlsync_set_extended_options, 219
 sp_hook_dbmlsync_set_ml_connect_info, 220
 sp_hook_dbmlsync_set_upload_end_progress, 222
 sp_hook_dbmlsync_sql_error, 224
 sp_hook_dbmlsync_upload_begin, 226
 sp_hook_dbmlsync_upload_end, 227
 sp_hook_dbmlsync_validate_download_file, 229
synchronization event hook sequence, 178

synchronization event hooks, 178
host protocol option
 MobiLink client connection option, 38
hovering
 dbmlsync, 87
HoverRescanThreshold dbmlsync extended option
 about, 135
hrt dbmlsync extended option
 about, 135
HTTP
 MobiLink client options, 25
HTTP synchronization
 MobiLink client options, 25
http_buffer_responses protocol option
 MobiLink client connection option, 38
http_password protocol option
 MobiLink client connection option, 39
http_proxy_password protocol option
 MobiLink client connection option, 40
http_proxy_userid protocol option
 MobiLink client connection option, 41
http_userid protocol option
 MobiLink client connection option, 42
HTTPS
 MobiLink client options, 26
HTTPS synchronization
 MobiLink client options, 26

I

iAnywhere developer community
 newsgroups, viii
iAnywhere.MobiLink.Client namespace
 Dbmlsync .NET API reference, 253
identity option
 MobiLink client connection option, 42
identity_password option
 MobiLink client connection option, 44
IDs
 MobiLink remote IDs, 9
IgnoreHookErrors
 MobiLink synchronization profile option, 158
IgnoreHookErrors dbmlsync extended option
 about, 136
IgnoreScheduling
 MobiLink synchronization profile option, 158
IgnoreScheduling dbmlsync extended option
 about, 136

- ignoring errors in event-hook procedures
 - SQL Anywhere clients, 183
- IMAP authentication
 - MobiLink scripts, 15
- inc dbmlsync extended option
 - about, 137
- Increment dbmlsync extended option
 - about, 137
- incremental uploads
 - MobiLink synchronization, 137
- InfoMessageEnabled property
 - dbmlsync integration component, 282
- Init method
 - DbmlsyncClient class [Dbmlsync .NET API], 261
 - DbmlsyncClient class [Dbmlsync C++ API], 239
- initiating
 - synchronization for SQL Anywhere clients, 79
- initiating synchronization
 - SQL Anywhere clients, 79
- initiating synchronization from an application
 - SQL Anywhere clients, 83
- install-dir
 - documentation usage, vi
- installing
 - MobiLink provider for Microsoft ActiveSync for SQL Anywhere clients, 84
- installing the MobiLink provider for Microsoft ActiveSync
 - SQL Anywhere clients, 84
- InstantiateClient method
 - DbmlsyncClient class [Dbmlsync .NET API], 262
 - DbmlsyncClient class [Dbmlsync C++ API], 240
- integration component
 - dbmlsync, 276
- interfaces
 - DBTools for dbmlsync, 304
- IRowTransferData interface
 - dbmlsync integration component, 300
- isc dbmlsync extended option
 - about, 136
- J**
- Java
 - MobiLink user authentication, 15
- Java and .NET user authentication
 - MobiLink, 15
- java.naming.provider.url
 - MobiLink external authenticator properties, 17
- K**
- KillConnections
 - MobiLink synchronization profile option, 158
- L**
- LDAP authentication
 - MobiLink scripts, 15
- locking
 - MobiLink SQL Anywhere clients, 82
- LockTables dbmlsync extended option
 - about, 138
 - concurrency during synchronization, 82
- log files
 - MobiLink SQL Anywhere client utility (dbmlsync) transaction logs, 81
 - MobiLink SQL Anywhere clients, 90
- log offsets
 - MobiLink SQL Anywhere clients, 66
- logging
 - MobiLink RI violations, 196
 - MobiLink SQL Anywhere client utility (dbmlsync) -v option, 123
 - MobiLink SQL Anywhere client utility (dbmlsync) transaction logs, 81
 - MobiLink SQL Anywhere client utility (dbmlsync)actions, 90
- logging dbmlsync activity
 - about, 90
- LogRenameSize
 - MobiLink synchronization profile option, 158
- logscan polling
 - about, 131
- lt dbmlsync extended option
 - about, 138
- M**
- Mac OS X
 - MobiLink, 91
- mail.imap.host
 - MobiLink external authenticator properties, 17
- mail.imap.port
 - MobiLink external authenticator properties, 17
- mail.pop3.host
 - MobiLink external authenticator properties, 17
- mail.pop3.port
 - MobiLink external authenticator properties, 17

- MobiLink external authenticator properties, 17
- Message event
 - dbmlsync integration component, 294
- message log
 - MobiLink SQL Anywhere client utility (dbmlsync) about, 90
- message log file
 - MobiLink SQL Anywhere client utility (dbmlsync) -o option, 111
 - MobiLink SQL Anywhere client utility (dbmlsync) -os option, 111
 - MobiLink SQL Anywhere client utility (dbmlsync) -ot option, 112
- Microsoft ActiveSync
 - class name for dbmlsync, 124
 - CREATE SYNCHRONIZATION USER statement for MobiLink SQL Anywhere clients, 84
 - installing the MobiLink provider, 18
 - installing the MobiLink provider for SQL Anywhere clients, 84
 - MobiLink Microsoft ActiveSync provider installation utility (mlasinst), 18
 - MobiLink SQL Anywhere clients, 83
 - registering applications for SQL Anywhere clients, 86
- Microsoft ActiveSync provider installation utility (mlasinst)
 - syntax, 18
- mirror logs
 - deleting for dbmlsync, 139
- MirrorLogDirectory dbmlsync extended option
 - about, 139
- ml_remote_id option
 - SQL Anywhere clients, 65
- ml_user
 - installing a SQL Anywhere client over an old one, 66
- ml_username
 - about, 4
 - creating, 5
- mlasdesk.dll
 - installing, 18
- mlasdev.dll
 - installing, 18
- mlasinst utility
 - dbmlsync usage, 83
 - installing the MobiLink provider for Microsoft ActiveSync for SQL Anywhere clients, 84
 - options, 18
 - syntax, 18
- mld dbmlsync extended option
 - about, 139
- mlfiletransfer utility
 - options, 21
 - syntax, 21
- MLUser
 - MobiLink synchronization profile option, 158
- mluser utility
 - using, 6
- mn dbmlsync extended option
 - about, 140
- Mobile Device Center (*see* Microsoft ActiveSync)
- MobiLink
 - connection parameters for clients, 23
 - dbmlsync event hooks, 178
 - dbmlsync options, 93
 - hooks, 178
 - logging RI violations, 196
 - scheduling SQL Anywhere clients, 87
 - scripted upload, 310
 - SQL Anywhere clients, 63
 - users, 4
 - utilities for clients, 18
- MobiLink client network protocol options
 - about, 23
- MobiLink client utilities
 - about, 18
- MobiLink client utility (dbmlsync)
 - options, 93
- MobiLink file transfer utility (mlfiletransfer)
 - syntax, 21
- MobiLink Microsoft ActiveSync provider installation utility (mlasinst)
 - syntax, 18
- MobiLink performance
 - estimate number of upload rows, 123
- MobiLink security
 - changing passwords, 8
 - choosing a user authentication mechanism, 11
 - custom user authentication, 14
 - new users, 7
 - passwords, 6
 - user authentication, 4
 - user authentication architecture, 12
 - user authentication passwords, 7
- MobiLink server

- Mac OS X, 91
- MobiLink SQL statements
 - listed, 156
- MobiLink synchronization
 - scheduling SQL Anywhere clients, 87
 - scripted upload, 310
 - SQL Anywhere clients, 63
- MobiLink synchronization client
 - options, 93
- MobiLink synchronization profiles
 - introduction, 157
 - options, 158
- MobiLink synchronization subscriptions
 - SQL Anywhere clients, 77
- MobiLink user names
 - about, 4
 - creating, 5
 - using in scripts, 10
- MobiLink users
 - about, 4
 - configuring properties at SQL Anywhere clients, 75
 - creating, 5
 - creating in SQL Anywhere clients, 74
 - dropping from SQL Anywhere clients, 76
- MobiLink utilities
 - client, 18
 - MobiLink file transfer (mlfiletransfer) syntax, 21
 - MobiLink Microsoft ActiveSync provider (mlasinst) syntax, 18
 - MobiLink SQL Anywhere client (dbmsync) syntax, 93
- MobiLinkPwd
 - MobiLink synchronization profile option, 158
- MobiLinkPwd dbmsync extended option
 - about, 140
- monitoring
 - logging MobiLink RI violations, 196
- mp dbmsync extended option
 - about, 140
- MSGQ_SHUTDOWN_REQUESTED
 - DBTools interface for dbmsync, 308
- MSGQ_SLEEP_THROUGH
 - DBTools interface for dbmsync, 308
- MSGQ_SYNC_REQUESTED
 - DBTools interface for dbmsync, 308

N

- name option
 - MobiLink Microsoft ActiveSync provider utility (mlasinst), 18
- named parameters
 - remote_id, 10
 - username, 10
- network parameters
 - MobiLink clients, 23
- network protocol options
 - dbmsync, 128
 - MobiLink clients, 23
- network protocols
 - MobiLink client options for HTTP, 25
 - MobiLink client options for HTTPS, 26
 - MobiLink client options for TCP/IP, 23
 - MobiLink client options for TLS, 24
 - specifying for dbmsync, 129
 - specifying for MobiLink, 2
 - UltraLite support for, 28
- network_adapter_name protocol option
 - MobiLink client connection option, 44
- network_leave_open protocol option
 - MobiLink client connection option, 45
- network_name protocol option
 - MobiLink client connection option, 46
- new users
 - MobiLink user authentication, 7
- NewMobiLinkPwd
 - MobiLink synchronization profile option, 158
- NewMobiLinkPwd dbmsync extended option
 - about, 140
- newsgroups
 - technical support, viii
- NoSyncOnStartup dbmsync extended option
 - about, 141
- nss dbmsync extended option
 - about, 141

O

- OfflineDirectory dbmsync extended option
 - about, 142
- offsets
 - MobiLink SQL Anywhere clients, 66
- online books
 - PDF, v
- operating systems

-
- Unix, v
 - Windows, v
 - Windows CE, v
 - Windows Mobile, v
 - options
 - MobiLink dbmlsync extended options, 126
 - MobiLink file transfer utility (mlfiletransfer), 21
 - MobiLink Microsoft ActiveSync provider utility (mlasinst), 18
 - MobiLink SQL Anywhere client utility (dbmlsync), 93
 - UltraLite network protocols, 28
 - options for performance tuning
 - MobiLink SQL Anywhere clients, 80
 - order of tables
 - dbmlsync extended option, 148
 - P**
 - p dbmlsync extended option
 - about, 131
 - Palm OS
 - version 12, deprecated feature, MobiLink, 1
 - parameters
 - MobiLink client connection, 23
 - partitioning
 - column-wise for MobiLink SQL Anywhere clients, 69
 - row-wise partitioning for MobiLink SQL Anywhere clients, 70
 - passthrough mode
 - (*see also* SQL passthrough)
 - passwords
 - changing for MobiLink, 8
 - MobiLink authentication by end users, 7
 - MobiLink user authentication setup, 6
 - path property
 - dbmlsync integration component, 279
 - PDF
 - documentation, v
 - performance
 - MobiLink SQL Anywhere clients, 80
 - persistent connections
 - dbmlsync -pc option, 113
 - persistent protocol option
 - MobiLink client connection option, 47
 - ping
 - dbmlsync synchronization parameter, 114
 - MobiLink synchronization profile option, 158
 - Ping method
 - DbmlsyncClient class [Dbmlsync .NET API], 262
 - DbmlsyncClient class [Dbmlsync C++ API], 240
 - pinging
 - MobiLink server, 114
 - polling
 - dbmlsync logscan polling, 131
 - PollingPeriod dbmlsync extended option
 - about, 142
 - POP3 authentication
 - MobiLink scripts, 15
 - port protocol option
 - MobiLink client connection option, 47
 - pp dbmlsync extended option
 - about, 142
 - preparing
 - remote databases for MobiLink, 63
 - priority order for extended options and connection parameters
 - SQL Anywhere clients, 126
 - procedures
 - MobiLink dbmlsync event hooks, 178
 - programming interfaces
 - dbmlsync, 89
 - progress
 - scripted upload, 318
 - progress offsets
 - MobiLink SQL Anywhere clients, 66
 - ProgressIndex event
 - dbmlsync integration component, 296
 - ProgressMessage event
 - dbmlsync integration component, 296
 - properties
 - dbmlsync integration component, 279
 - protocol options
 - dbmlsync, 128
 - MobiLink clients, 23
 - protocols
 - (*see also* network protocols)
 - MobiLink client options for HTTP, 25
 - MobiLink client options for HTTPS, 26
 - MobiLink client options for TCP/IP, 23
 - MobiLink client options for TLS, 24
 - specifying for dbmlsync, 129
 - UltraLite list, 28
 - proxy_host protocol option
 - MobiLink client connection option, 48

- proxy_hostname option
 - MobiLink client connection option, 48
- proxy_port protocol option
 - MobiLink client connection option, 49
- proxy_portnumber option
 - MobiLink client connection option, 49
- Publication
 - MobiLink synchronization profile option, 158
- publications
 - about MobiLink SQL Anywhere clients, 67
 - altering for MobiLink SQL Anywhere clients, 72
 - column-wise partitioning for MobiLink SQL Anywhere clients, 69
 - creating for MobiLink SQL Anywhere clients, 67
 - download-only, 71
 - dropping from MobiLink SQL Anywhere clients, 74
 - MobiLink SQL Anywhere client offsets, 66
 - row-wise partitioning for MobiLink SQL Anywhere clients, 70
 - simple publications for MobiLink SQL Anywhere clients, 68
 - using a WHERE clause in MobiLink, 70
- publishing
 - MobiLink selected columns (SQL Anywhere clients), 69
 - MobiLink selected rows (SQL Anywhere clients), 70
 - MobiLink whole tables (SQL Anywhere clients), 68
 - selected columns for MobiLink SQL Anywhere clients, 69
 - selected rows in MobiLink, 70
 - SQL Anywhere client tables, 67
 - SQL Anywhere client whole tables, 68
- publishing data
 - MobiLink SQL Anywhere clients, 67
- R**
- reconciling data (*see* synchronization)
- referential integrity
 - resolving MobiLink RI violations, 196
- registering
 - MobiLink SQL Anywhere applications with Microsoft ActiveSync, 86
 - MobiLink users, 5
- registering MobiLink users
 - about, 5
- remote databases
 - creating SQL Anywhere clients, 63
 - deploying SQL Anywhere clients, 63
 - MobiLink SQL Anywhere clients, 63
 - restoring from backup, 116
 - transferring files, 21
- remote DBA permissions
 - MobiLink synchronization of SQL Anywhere clients, 80
- remote IDs
 - about, 9
 - setting in SQL Anywhere databases, 65
- remote MobiLink databases
 - schema changes, 58
- RemoteProgressGreater
 - MobiLink synchronization profile option, 158
- RemoteProgressLess
 - MobiLink synchronization profile option, 158
- removing
 - articles from MobiLink SQL Anywhere clients, 72
- restartable downloads
 - dbmsync -dc option, 103
 - sp_hook_dbmsync_end, 203
- restoring
 - remote databases from backup, 116
- return codes
 - dbmsync [sp_hook_dbmsync_abort], 183
 - dbmsync [sp_hook_dbmsync_process_exit_code], 216
- RI violations
 - MobiLink dbmsync clients, 182
- ROLLBACK statement
 - event-hook procedures, 180
- row-wise partitioning
 - MobiLink SQL Anywhere clients, 70
- RowOperation property
 - dbmsync integration component, 301
- RSA
 - MobiLink clients, 51
- RSA protocol option
 - MobiLink clients, 51
- run method
 - dbmsync integration component, 278
- S**
- s.remote_id

- usage, 10
- s.username
 - MobiLink use in scripts, 10
- sa dbmsync extended option
 - about, 146
- samples-dir
 - documentation usage, vi
- sch dbmsync extended option
 - about, 143
- Schedule dbmsync extended option
 - about, 143
- schedules
 - MobiLink SQL Anywhere clients, 87
- scheduling
 - ignore for dbmsync, 136
 - MobiLink SQL Anywhere client utility (dbmsync)
 - Schedule extended option, 143
 - MobiLink SQL Anywhere clients, 87
 - MobiLink using sp_hook_dbmsync_delay, 191
 - MobiLink using sp_hook_dbmsync_end, 203
- scheduling synchronization
 - SQL Anywhere clients, 87
- schema changes
 - remote MobiLink databases, 58
- schema upgrades
 - sp_hook_dbmsync_schema_upgrade event hook, 217
 - SQL Anywhere remote databases, 60
 - UltraLite remote databases, 61
- scn dbmsync extended option
 - about, 146
- script parameters
 - remote_id, 10
 - username, 10
- script-based upload
 - about MobiLink, 310
- scripted upload
 - about MobiLink, 310
 - MobiLink custom progress values, 319
 - MobiLink defining stored procedures for inserts, 320
 - MobiLink defining stored procedures for scripted upload, 318
 - MobiLink design, 312
 - MobiLink example, 323
- scripted uploads
 - MobiLink defining stored procedures for deletes, 321
 - MobiLink defining stored procedures for updates, 322
- scripts
 - MobiLink remote_id parameter, 10
- ScriptVersion dbmsync extended option
 - about, 145
- security
 - changing MobiLink passwords, 8
 - MobiLink custom user authentication, 14
 - MobiLink new users, 7
 - MobiLink synchronization of SQL Anywhere clients, 80
 - MobiLink user authentication, 4
 - user authentication passwords, 7
- selecting
 - UltraLite network protocols, 28
- send column names
 - dbmsync extended option, 146
- send download acknowledgement
 - dbmsync extended option, 146
- SendColumnNames dbmsync extended option
 - about, 146
- SendDownloadAck dbmsync extended option
 - about, 146
- SendTriggers dbmsync extended option
 - about, 147
- sequences
 - synchronization event hooks, 178
- server stored procedures
 - MobiLink dbmsync event hooks, 178
- set_cookie protocol option
 - MobiLink client connection option, 49
- SetProperty method
 - DbmsyncClient class [Dbmsync .NET API], 262
 - DbmsyncClient class [Dbmsync C++ API], 241
- setting remote IDs
 - SQL Anywhere databases, 65
- setting up
 - MobiLink DBTools interface for dbmsync, 305
 - MobiLink scripted upload, 311
- setting up the dbmsync integration component
 - about, 277
- SetTitle event
 - dbmsync integration component, 297
- ShutdownServer method
 - DbmsyncClient class [Dbmsync .NET API], 264
 - DbmsyncClient class [Dbmsync C++ API], 242
- shutting down

- dbmsync automatically, 116
- sp_hook_dbmsync_abort
 - syntax, 183
- sp_hook_dbmsync_all_error
 - syntax, 185
- sp_hook_dbmsync_begin
 - syntax, 188
- sp_hook_dbmsync_communication_error
 - syntax, 189
- sp_hook_dbmsync_delay
 - syntax, 191
- sp_hook_dbmsync_download_begin
 - syntax, 194
- sp_hook_dbmsync_download_end
 - syntax, 195
- sp_hook_dbmsync_download_log_ri_violation
 - syntax, 196
- sp_hook_dbmsync_download_ri_violation
 - syntax, 198
- sp_hook_dbmsync_download_table_begin
 - syntax, 200
- sp_hook_dbmsync_download_table_end
 - syntax, 201
- sp_hook_dbmsync_end
 - syntax, 203
- sp_hook_dbmsync_log_rescan
 - syntax, 205
- sp_hook_dbmsync_logscan_begin
 - syntax, 207
- sp_hook_dbmsync_logscan_end
 - syntax, 208
- sp_hook_dbmsync_misc_error
 - syntax, 210
- sp_hook_dbmsync_ml_connect_failed
 - syntax, 213
- sp_hook_dbmsync_process_exit_code
 - syntax, 216
- sp_hook_dbmsync_schema_upgrade
 - syntax, 217
- sp_hook_dbmsync_set_extended_options
 - syntax, 219
- sp_hook_dbmsync_set_ml_connect_info
 - syntax, 220
- sp_hook_dbmsync_set_upload_end_progress
 - syntax, 222
- sp_hook_dbmsync_sql_error
 - syntax, 224
- sp_hook_dbmsync_upload_begin
 - syntax, 226
- sp_hook_dbmsync_upload_end
 - syntax, 227
- sp_hook_dbmsync_validate_download_file
 - syntax, 229
- specifying the network protocol for clients
 - MobiLink, 2
- SQL Anywhere
 - as MobiLink clients, 1
 - documentation, v
- SQL Anywhere client logging
 - about, 90
- SQL Anywhere client utility (dbmsync)
 - syntax, 93
- SQL Anywhere clients
 - about MobiLink, 63
 - dbmsync integration component, 276
 - introduction, 1
 - MobiLink SQL Anywhere client utility (dbmsync), 93
 - registering for Microsoft ActiveSync, 86
- SQL Anywhere Developer Centers
 - finding out more and requesting technical support, ix
- SQL Anywhere remote databases
 - about MobiLink, 63
- SQL Anywhere Tech Corner
 - finding out more and requesting technical support, ix
- SQL statements
 - MobiLink, 156
- src option
 - MobiLink Microsoft ActiveSync provider utility (mlasinst), 18
- st dbmsync extended option
 - about, 147
- StartServer method
 - DbmsyncClient class [Dbmsync .NET API], 265
 - DbmsyncClient class [Dbmsync C++ API], 243
- state
 - MobiLink SQL Anywhere clients, 66
- statements
 - MobiLink, 156
- stop method
 - dbmsync integration component, 278
- stored procedures
 - MobiLink client procedures, 178
 - MobiLink dbmsync event hooks, 178

- sp_hook_dbmsync_abort syntax, 183
- sp_hook_dbmsync_all_error syntax, 185
- sp_hook_dbmsync_begin syntax, 188
- sp_hook_dbmsync_communication_error syntax, 189
- sp_hook_dbmsync_delay syntax, 191
- sp_hook_dbmsync_download_begin syntax, 194
- sp_hook_dbmsync_download_end syntax, 195
- sp_hook_dbmsync_download_log_ri_violation, 196
- sp_hook_dbmsync_download_ri_violation, 198
- sp_hook_dbmsync_download_table_begin syntax, 200
- sp_hook_dbmsync_download_table_end syntax, 201
- sp_hook_dbmsync_end syntax, 203
- sp_hook_dbmsync_log_rescan syntax, 205
- sp_hook_dbmsync_logscan_begin syntax, 207
- sp_hook_dbmsync_logscan_end syntax, 208
- sp_hook_dbmsync_misc_error syntax, 210
- sp_hook_dbmsync_ml_connect_failed syntax, 213
- sp_hook_dbmsync_process_exit_code syntax, 216
- sp_hook_dbmsync_schema_upgrade syntax, 217
- sp_hook_dbmsync_set_extended_options syntax, 219
- sp_hook_dbmsync_set_ml_connect_info syntax, 220
- sp_hook_dbmsync_set_upload_end_progress syntax, 222
- sp_hook_dbmsync_sql_error syntax, 224
- sp_hook_dbmsync_upload_begin syntax, 226
- sp_hook_dbmsync_upload_end syntax, 227
- sp_hook_dbmsync_validate_download_file syntax, 229
- stream parameters
 - (*see also* protocol options)
 - MobiLink clients, 23
- Subscription
 - MobiLink synchronization profile option, 158
- subscriptions
 - MobiLink SQL Anywhere clients, 77
- support
 - newsgroups, viii
- supported network protocols
 - UltraLite list, 28
- supported platforms
 - dbmsync integration component, 277
- sv dbmsync extended option
 - about, 145
 - switches
 - MobiLink file transfer utility (mlfiletransfer), 21
 - MobiLink Microsoft ActiveSync provider utility (mlasinst), 18
 - MobiLink SQL Anywhere client utility (dbmsync), 93
 - Sync method
 - DbmsyncClient class [Dbmsync .NET API], 266
 - DbmsyncClient class [Dbmsync C++ API], 244
 - SyncConsole
 - getting started, 91
 - synchronization
 - changing passwords, 8
 - connection parameters for clients, 23
 - custom user authentication, 14
 - customizing, 178
 - dbmsync first synchronization, 66
 - initiating for SQL Anywhere clients, 79
 - Microsoft ActiveSync for MobiLink SQL Anywhere clients, 83
 - MobiLink client utilities, 18
 - MobiLink dbmsync event hooks, 178
 - scheduling dbmsync, 143
 - scheduling MobiLink SQL Anywhere clients, 87
 - SQL Anywhere clients, 63
 - transactions, 180
 - synchronization event hook sequence
 - SQL Anywhere clients, 178
 - synchronization profiles
 - sp option, 119
 - SQL Anywhere clients, 157
 - synchronization subscriptions
 - (*see also* subscriptions)
 - altering for SQL Anywhere clients, 78
 - dropping from SQL Anywhere clients, 78
 - priority order for extended options and connection parameters, 126
 - SQL Anywhere clients, 77
 - synchronization users
 - about, 4
 - configuring properties at SQL Anywhere clients, 75
 - creating, 5
 - creating in SQL Anywhere clients, 74
 - dropping from SQL Anywhere clients, 76
 - synchronizing
 - (*see also* synchronization)

- MobiLink new users, 7
- syntax
 - MobiLink dbmsync event hooks, 178
 - MobiLink file transfer utility (mlfiletransfer), 21
 - MobiLink Microsoft ActiveSync provider installation utility (mlasinst), 18
 - MobiLink SQL Anywhere client utility (dbmsync), 93
 - MobiLink synchronization utilities for clients, 18
- system procedures
 - MobiLink dbmsync event hooks, 178
- system tables
 - MobiLink clients, 3

T

- table order
 - dbmsync extended option, 148
- TableName property
 - dbmsync integration component, 301
- TableOrder dbmsync extended option
 - about, 148
- TableOrderChecking dbmsync extended option
 - about, 149
- tables
 - adding to remote MobiLink SQL Anywhere databases, 60
 - column-wise partitioning for MobiLink SQL Anywhere clients, 69
 - publishing for MobiLink SQL Anywhere clients, 67
 - row-wise partitioning for MobiLink SQL Anywhere clients, 70
- TCP/IP
 - MobiLink client options, 23
 - MobiLink client options for TLS, 24
- TCP/IP synchronization
 - MobiLink client options, 23
 - MobiLink client options for TLS, 24
- tech corners
 - finding out more and requesting technical support, ix
- technical support
 - newsgroups, viii
- timeout protocol option
 - MobiLink client connection option, 50
- TLS
 - MobiLink client options, 24

- TLS synchronization
 - MobiLink client options, 24
- tls_type protocol option
 - MobiLink client connection option, 51
- toc dbmsync extended option
 - about, 149
- tor dbmsync extended option
 - about, 148
- transaction log
 - mirror log deletion for dbmsync, 139
 - MobiLink SQL Anywhere client utility (dbmsync), 81
- transaction log files
 - MobiLink SQL Anywhere client utility (dbmsync), 81
- transaction log mirror
 - deleting for dbmsync, 139
- transaction-level uploads
 - dbmsync -tu option, 119
- transactional uploads (*see* transaction-level uploads)
- TransactionalUpload
 - MobiLink synchronization profile option, 158
- transferring files
 - MobiLink file transfer utility (mlfiletransfer), 21
- troubleshooting
 - MobiLink dbmsync log, 90
 - MobiLink deployment of SQL Anywhere clients, 66
 - newsgroups, viii
 - restoring the remote database from backup, 116
- trusted_certificates protocol option
 - MobiLink client connection option, 53

U

- UltraLite
 - MobiLink clients, 1
- UltraLite applications
 - as MobiLink clients, 1
- UltraLite clients
 - introduction, 1
- UltraLite network protocols
 - compression deployment requirements, 28
 - synchronization options for HTTP, 28
 - synchronization options for HTTPS, 28
 - synchronization options for TCP/IP, 28
 - synchronization options for TLS, 28
- UltraLite protocols

- synchronization options for HTTP, 28
- synchronization options for HTTPS, 28
- synchronization options for TCP/IP, 28
- synchronization options for TLS, 28
- UltraLite synchronization
 - compressed synchronization deployment requirements, 28
 - HTTP client options, 28
 - HTTPS client options, 28
 - TCP/IP client options, 28
 - TLS client options, 28
- UltraLiteJ
 - MobiLink clients, 2
- UltraLiteJ clients
 - introduction, 2
- Unix
 - documentation conventions, v
 - operating systems, v
- uo dbmlsync extended option
 - about, 150
- UpdateGenNum
 - MobiLink synchronization profile option, 158
- upgrading
 - schemas in MobiLink remote databases, 58
- upgrading remote databases
 - MobiLink SQL Anywhere clients, 66
- UPLD_ERR_INVALID_USERID_OR_PASSWORD
 - dbmlsync error message, 228
- UPLD_ERR_REMOTE_ID_ALREADY_IN_USE
 - dbmlsync error message, 228
- UPLD_ERR_SQLCODE_n
 - dbmlsync error message, 228
- UPLD_ERR_USERID_OR_PASSWORD_EXPIRED
 - dbmlsync error message, 228
- upload only synchronization
 - (*see also* upload-only synchronization)
- upload-only synchronization
 - dbmlsync -uo option, 122
 - SQL Anywhere remote databases, 150
- UploadAck event
 - dbmlsync integration component, 298
- UploadEventsEnabled property
 - dbmlsync integration component, 280
- uploading data
 - scripted upload in MobiLink, 310
- UploadOnly
 - MobiLink synchronization profile option, 158
- UploadOnly dbmlsync extended option
 - about, 150
- UploadRow event
 - dbmlsync integration component, 299
- UploadRowCnt
 - MobiLink synchronization profile option, 158
- uploads
 - MobiLink scripted upload, 310
 - MobiLink SQL Anywhere client utility (dbmlsync)
 - uo option for upload-only synchronization, 122
- url_suffix protocol option
 - MobiLink client connection option, 54
- UseNaturalTypes property
 - dbmlsync integration component, 283
- user authentication
 - .NET synchronization logic, 15
 - changing MobiLink passwords, 8
 - choosing a mechanism in MobiLink, 11
 - Java synchronization logic, 15
 - MobiLink architecture, 12
 - MobiLink custom mechanism, 14
 - MobiLink new users, 7
 - MobiLink passwords, 6
 - MobiLink security, 4
 - passwords, 7
- user authentication architecture
 - MobiLink, 12
- user names
 - MobiLink about, 4
 - MobiLink creating, 5
 - MobiLink use in scripts, 10
- username
 - MobiLink about, 4
 - MobiLink creating, 5
 - MobiLink use in scripts, 10
- users
 - MobiLink about, 4
 - MobiLink creating, 5
 - MobiLink creation in SQL Anywhere clients, 74
- utilities
 - MobiLink file transfer (mlfiletransfer) syntax, 21
 - MobiLink list of client utilities, 18
 - MobiLink Microsoft ActiveSync provider (mlasinst) syntax, 18
 - MobiLink SQL Anywhere client (dbmlsync) syntax, 93

V

- v dbmsync extended option
 - about, 150
- Verbose dbmsync extended option
 - about, 150
- VerboseHooks dbmsync extended option
 - about, 151
- VerboseMin dbmsync extended option
 - about, 152
- VerboseOptions dbmsync extended option
 - about, 153
- VerboseRowCounts dbmsync extended option
 - about, 154
- VerboseRowValues dbmsync extended option
 - about, 155
- VerboseUpload dbmsync extended option
 - about, 156
- Verbosity
 - MobiLink synchronization profile option, 158
- verbosity
 - MobiLink SQL Anywhere client utility (dbmsync) setting, 123
- verbosity option
 - MobiLink SQL Anywhere client utility (dbmsync), 123
- verifying certificate fields
 - MobiLink TLS certificate_company option, 29
 - MobiLink TLS certificate_name option, 30
 - MobiLink TLS certificate_unit option, 32
- version protocol option
 - MobiLink client connection option, 55
- vm dbmsync extended option
 - about, 152
- vn dbmsync extended option
 - about, 154
- vo dbmsync extended option
 - about, 153
- vr dbmsync extended option
 - about, 155
- vs dbmsync extended option
 - about, 151
- vu dbmsync extended option
 - about, 156

W

- WaitForServerShutdown method
 - DbmsyncClient class [Dbmsync .NET API], 267

- DbmsyncClient class [Dbmsync C++ API], 245
- WaitingForUploadAck event
 - dbmsync integration component, 300
- WarningMessageEnabled property
 - dbmsync integration component, 281
- WHERE clause
 - MobiLink publications, 70
- Windows
 - documentation conventions, v
 - operating systems, v
- Windows CE (*see* Windows Mobile)
- Windows Mobile
 - dbmsync preloading DLLs, 114
 - documentation conventions, v
 - operating systems, v
 - Windows CE, v
- Windows Mobile Device Center (*see* Microsoft ActiveSync)

Z

- zlib compression
 - MobiLink synchronization, 33
- zlib_download_window_size protocol option
 - MobiLink client connection option, 56
- zlib_upload_window_size protocol option
 - MobiLink client connection option, 57