MobiLink
Client Administration

February 2009

Version 11.0.1

Copyright and trademarks

Contents

About this book

Subject

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.

Audience

This book is for users who want to add synchronization to their information systems.

Before you begin

For a comparison of MobiLink with other synchronization and replication technologies, see "Overview of data exchange technologies" [*SQL Anywhere 11 - Introduction*].

About the SQL Anywhere documentation

The complete SQL Anywhere documentation is available in four formats that contain identical information.

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

  If you are using a Microsoft Windows operating system, the online Help is provided in HTML Help (CHM) format. To access the documentation, choose **Start** » **Programs** » **SQL Anywhere 11** » **Documentation** » **Online Books**.

  The administration tools use the same online documentation for their Help features.

- **Eclipse** On Unix platforms, the complete online Help is provided in Eclipse format. To access the documentation, run *sadoc* from the *bin32* or *bin64* directory of your SQL Anywhere 11 installation.

- **DocCommentXchange** DocCommentXchange 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

  Visit http://dcx.sybase.com.

- **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 download Adobe Reader, visit http://get.adobe.com/reader/.

  To access the PDF documentation on Microsoft Windows operating systems, choose **Start** » **Programs** » **SQL Anywhere 11** » **Documentation** » **Online Books - PDF Format**.

  To access the PDF documentation on Unix operating systems, use a web browser to open *install-dir/documentation/en/pdf/index.html*.

About the books in the documentation set

The SQL Anywhere documentation consists of the following books:

- **SQL Anywhere 11 - Introduction** This book introduces SQL Anywhere 11, a comprehensive package that provides data management and data exchange, enabling the rapid development of database-powered applications for server, desktop, mobile, and remote office environments.

- **SQL Anywhere 11 - Changes and Upgrading** This book describes new features in SQL Anywhere 11 and in previous versions of the software.

- **SQL Anywhere Server - Database Administration** This book describes how to run, manage, and configure SQL Anywhere databases. It describes database connections, the database server, database files, backup procedures, security, high availability, replication with the Replication Server, and administration utilities and options.

- **SQL Anywhere Server - Programming** This book describes how to build and deploy database applications using the C, C++, Java, PHP, Perl, Python, and .NET programming languages such as Visual Basic and Visual C#. A variety of programming interfaces such as ADO.NET and ODBC are described.
- **SQL Anywhere Server - SQL Reference** This book provides reference information for system procedures, and the catalog (system tables and views). It also provides an explanation of the SQL Anywhere implementation of the SQL language (search conditions, syntax, data types, and functions).
- **SQL Anywhere Server - SQL Usage** This book describes how to design and create databases; how to import, export, and modify data; how to retrieve data; and how to build stored procedures and triggers.
- **MobiLink - Getting Started** This book introduces MobiLink, a session-based relational-database synchronization system. MobiLink technology allows two-way replication and is well suited to mobile computing environments.
- **MobiLink - Client Administration** 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.
- **MobiLink - Server Administration** This book describes how to set up and administer MobiLink applications.
- **MobiLink - Server-Initiated Synchronization** This book describes MobiLink server-initiated synchronization, a feature that allows the MobiLink server to initiate synchronization or perform actions on remote devices.
- **QAnywhere** This book describes QAnywhere, which is a messaging platform for mobile, wireless, desktop, and laptop clients.
- **SQL Remote** This book describes the SQL Remote data replication system for mobile computing, which enables sharing of data between a SQL Anywhere consolidated database and many SQL Anywhere remote databases using an indirect link such as email or file transfer.
- **UltraLite - Database Management and Reference** This book introduces the UltraLite database system for small devices.
- **UltraLite - C and C++ Programming** This book describes UltraLite C and C++ programming interfaces. With UltraLite, you can develop and deploy database applications to handheld, mobile, or embedded devices.
- **UltraLite - M-Business Anywhere Programming** This book describes UltraLite for M-Business Anywhere. With UltraLite for M-Business Anywhere you can develop and deploy web-based database applications to handheld, mobile, or embedded devices, running Palm OS, Windows Mobile, or Windows.
- **UltraLite - .NET Programming** This book describes UltraLite.NET. With UltraLite.NET you can develop and deploy database applications to computers, or handheld, mobile, or embedded devices.
- **UltraLiteJ** This book describes UltraLiteJ. With UltraLiteJ, you can develop and deploy database applications in environments that support Java. UltraLiteJ supports BlackBerry smartphones and Java SE environments. UltraLiteJ is based on the iAnywhere UltraLite database product.

- **Error Messages** This book provides a complete listing of SQL Anywhere error messages together with diagnostic information.

Documentation conventions

This section lists the conventions used in this documentation.

Operating systems

SQL Anywhere runs on a variety of platforms. In most cases, the software behaves 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 (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 Windows Vista and Windows XP, used primarily on server, desktop, and laptop computers, and Windows Mobile used on mobile devices.

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

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

Directory and file names

In most cases, 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. In most cases, you can 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 *dbsrv11.exe*. On Unix, it is *dbsrv11*.

- ***install-dir*** During the installation process, you choose where to install SQL Anywhere. The environment variable SQLANY11 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 *%SQLANY11%\readme.txt*. On Unix, this is equivalent to *$SQLANY11/readme.txt* or *${SQLANY11}/readme.txt*.

  For more information about the default location of *install-dir*, see "SQLANY11 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 SQLANYSAMP11 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*, from the **Start** menu, choose **Programs** » **SQL Anywhere 11** » **Sample Applications And Projects**.

  For more information about the default location of *samples-dir*, see "SQLANYSAMP11 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)"
```

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

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

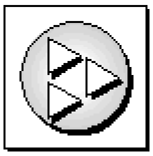
  In many shells, the value of the key would be `my "secret" key`.

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

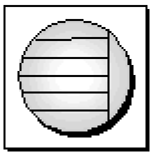
Graphic icons

The following icons are used in this documentation.

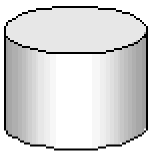
- A client application.



- A database server, such as Sybase SQL Anywhere.



- A database. In some high-level diagrams, the icon may be used to represent both the database and the database server that manages it.



- Replication or synchronization middleware. These assist in sharing data among databases. Examples are the MobiLink server and the SQL Remote Message Agent.

- A programming interface.

Interface

Contacting the documentation team

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

To submit your comments and suggestions, send an email to the SQL Anywhere documentation team at iasdoc@sybase.com. Although we do not reply to emails, your feedback helps us to improve our documentation, so your input is welcome.

DocCommentXchange

You can also 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

Visit http://dcx.sybase.com.

Finding out more and requesting technical support

Additional information and resources are available at the Sybase iAnywhere Developer Community at http://www.sybase.com/developer/library/sql-anywhere-techcorner.

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: **dbeng11 -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.

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

Contents

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 96
- "Publishing data" on page 100
- "Creating MobiLink users" on page 107
- "Creating synchronization subscriptions" on page 110

Synchronization is initiated by running a command line utility called dbmlsync. This utility connects to the remote database and prepares the upload, typically using information contained in the transaction log of the remote database. (Alternatively, you can implement scripted upload and not use the transaction log.) The dbmlsync utility can use information stored in a synchronization publication and synchronization subscription to connect to the MobiLink server and exchange data.

See "Initiating synchronization" on page 113.

For more information about SQL Anywhere clients, see "SQL Anywhere clients" on page 95.

For details of dbmlsync command line options, see "MobiLink SQL Anywhere client utility (dbmlsync)" on page 129.

Customizing synchronization

See "Customizing dbmlsync synchronization" on page 123.

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 ActiveSync, synchronization is initiated from your application by a single call to a synchronization function. The interface for HotSync is slightly different. Once synchronization is initiated from the application or from HotSync, the MobiLink server and the UltraLite runtime control the actions that occur during synchronization.

See also

- UltraLite - Database Management and Reference
- "UltraLite clients" [*UltraLite - Database Management and Reference*]
- "UltraLiteJ clients" on page 6
- "Using ActiveSync and HotSync with UltraLite" [*UltraLite - Database Management and Reference*]
- "UltraLite synchronization parameters and network protocol options" [*UltraLite - Database Management and Reference*]

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) and SQL passthrough feature are not available for UltraLiteJ clients.

See also

- “Data synchronization” [*UltraLiteJ*]
- “Using UltraLiteJ as a MobiLink client” [*UltraLiteJ*]
- UltraLite - Database Management and Reference
- “UltraLite clients” [*UltraLite - Database Management and Reference*]
- “UltraLite synchronization parameters and network protocol options” [*UltraLite - Database Management and Reference*]

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 mlsrv11 command line option is:

mlsrv11 -c "*connection-string***" -x** *protocol***(** *options* **)**

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

```
mlsrv11 -c "dsn=SQL Anywhere 11 Demo" -x tcpip
```

You can configure your protocol using options of the form:

(*keyword***=***value***;...)**

For example:

```
mlsrv11 -c "dsn=SQL Anywhere 11 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 option" [*MobiLink - Server Administration*]
All the network protocol options available to MobiLink client applications	"MobiLink client network protocol option summary" on page 35
How to set options for SQL Anywhere clients	"CommunicationAddress (adr) extended option" on page 187 "CommunicationType (ctp) extended option" on page 189
How to set options for UltraLite clients	"Stream Parameters synchronization parameter" [*UltraLite - Database Management and Reference*] "Stream Type synchronization parameter" [*UltraLite - Database Management and Reference*] "UltraLite Synchronization utility (ulsync)" [*UltraLite - Database Management and Reference*]

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*]
- "SYSSYNCSCRIPT 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

Contents

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.

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 15.

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 107.

- 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 mlsrv11. 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 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 Admin mode)

1. Connect to the consolidated database from Sybase Central using the MobiLink plug-in.
2. Choose **Mode** » **Admin**.
3. Click **Users**.
4. Choose **File** » **New** » **User**.
5. Follow the instructions in the **Create MobiLink 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 by default 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 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. If they do not, they are prompted for one in the dbmlsync connection window. The latter method is more secure 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 option" on page 142.

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 option" on page 158 and "-mn option" on page 157.

  - **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 11 and "Synchronizations from new users" on page 12.

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, MobiLink 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 until it is manually changed.

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.

The MobiLink server issues an error if the same remote ID is used in two or more concurrent synchronizations.

See also

- "ml_subscription" [*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 97.

For UltraLite clients, see "UltraLite ml_remote_id option" [*UltraLite - Database Management and Reference*].

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 pre-populate 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, and when the MobiLink user name uniquely identifies a remote database, 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 20.

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 10.

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 20.

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:

- mlsrv11 -zu+ option
- authenticate_user script or authenticate_user_hashed script
- authenticate_parameters script

The mlsrv11 -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 21.

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

See also

- "-zu 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 Model mode, using the Authentication tab. See "MobiLink models" [*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 23 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 ianywhere.ml.authentication.*;
import ianywhere.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

Contents

Introduction to MobiLink client utilities

There are two MobiLink client utilities:

- "ActiveSync provider installation utility (mlasinst)" on page 27
- "MobiLink file transfer utility (mlfiletransfer)" on page 30

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*]

ActiveSync provider installation utility (mlasinst)

Installs a MobiLink provider for 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.
-k *path*	Specify the location of the desktop provider *mlasdesk.dll*. By default, the file is located in *install-dir\bin32*. End users (who generally do not have the full SQL Anywhere install) may need to specify -k when installing the MobiLink ActiveSync provider.
-l *filename*	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 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 *n*	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 ActiveSync. This option unregisters all applications that have been registered for use with the MobiLink ActiveSync provider and uninstalls the MobiLink 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 *path*	Specify the location of the device provider *mlasdev.dll*. By default, the file is looked for in a platform-specific directory in *install-dir\CE*. End users (who generally do not have the full SQL Anywhere install) may need to specify -v when installing the MobiLink ActiveSync provider.

Other parameters	Description
src	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.
dst	Specify the destination file name and path on the device for an application.
name	Specify the name by which ActiveSync refers to the application.
class	Specify the registered Windows class name of the application.
args	Specify command line arguments to pass to the application when ActiveSync starts the application.

Remarks

This utility installs a MobiLink provider for 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 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 ActiveSync.

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

For complete instructions on using the ActiveSync provider installation utility, see:

- SQL Anywhere: "Installing the MobiLink provider for ActiveSync" on page 118
- UltraLite: "ActiveSync on Windows Mobile" [*UltraLite - Database Management and Reference*]

Examples

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

```
mlasinst -u
```

The following command installs the MobiLink provider for 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 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 ActiveSync synchronization" on page 117
- "UltraLite synchronization parameters and network protocol options" [*UltraLite - Database Management and Reference*]

MobiLink file transfer utility (mlfiletransfer)

Downloads a file through MobiLink.

Syntax

mlfiletransfer [*options*] *file*

Option	Description
-ap *param1, ...*	MobiLink authentication parameters. See "Authentication parameters" [*MobiLink - Server Administration*].
-dp *path*	The local path where the downloaded file is to be stored. By default, the downloaded file is stored in the root directory on Windows Mobile, and in the current directory on other platforms.
-df *filename*	The local name of the downloaded file. Use this option if you want to have a different name for the file on the client than the one used on the server. By default, the server name is used.
-f	Forces a download, even the local file is up to date. Any previous partial download is discarded.
-g	Shows download progress.
-p *password*	The password for the MobiLink user name.
-q	Quiet mode. Messages are not displayed.
-r	Enables download resumption. When set, the utility resumes a partial previous download that was interrupted because of a communication error or because it was canceled by the user. When the file on the server is newer than the partial file, the partial file is discarded. The -f option overrides this option.
-u *username*	MobiLink user name. This option is required.
-v *version*	The script version. This option is required.
-x *protocol***(***options***)**	The *protocol* can be one of **tcpip**, **tls**, **http**, or **https**.This option is required. 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 35.

Option	Description
file	The file to be transferred as named on the server. Do not include a path, as the location of the file is determined by the mlsrv11 -ftr option (which must be used when you start the MobiLink server). MobiLink looks for the file in the username subdirectory of the -ftr directory; if it doesn't find it there, it looks in the -ftr directory. If the file is not in either place, MobiLink generates an error. See "-ftr option" [*MobiLink - Server Administration*].

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, 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.

UltraLite users can also use the MLFileTransfer method in the UltraLite runtime. See "Using MobiLink file transfers" [*UltraLite - Database Management and Reference*].

See also

- "-ftr 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.

```
mlsrv11 -c "dsn=SQL Anywhere 11 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 10.0" -x tcpip notepad.exe
```

MobiLink client network protocol options

Contents

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 187
UltraLite	"Stream Parameters synchronization parameter" [*UltraLite - Database Management and Reference*] or -x option in "UltraLite Synchronization utility (ulsync)" [*UltraLite - Database Management and Reference*]
UltraLiteJ	"Network protocol options for UltraLiteJ synchronization streams" [*UltraLiteJ*]
Redirector	ML directive in "Configuring Redirector properties (for Redirectors that support server groups)" [*MobiLink - Server Administration*] or "Configuring Redirector properties (for Redirectors that don't support server groups)" [*MobiLink - Server Administration*]
MobiLink Monitor	"Starting the MobiLink Monitor" [*MobiLink - Server Administration*]
MobiLink file transfer	"MobiLink file transfer utility (mlfiletransfer)" on page 30
MobiLink Listener	-x in "Listener utility for Windows devices" [*MobiLink - Server-Initiated Synchronization*]
QAnywhere Agent	"-x option" [*QAnywhere*]

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 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...
buffer_size=*bytes*	"buffer_size" on page 40
client_port=*nnnnn*[-*mmmmm*]	"client_port" on page 46

TCP/IP protocol option	For more information, see...
e2ee_type={**rsa**|**ecc**}	"e2ee_type" on page 49
e2ee_public_key=*file*	"e2ee_public_key" on page 50
compression={**zlib**|**none**}	"compression" on page 47
host=*hostname*	"host" on page 53
network_leave_open={**off**|**on**}	"network_leave_open" on page 59
network_name=*name*	"network_name" on page 60
port=*portnumber*	"port" on page 63
timeout=*seconds*	"timeout" on page 67
zlib_download_window_size=*window-bits*	"zlib_download_window_size" on page 76
zlib_upload_window_size=*window-bits*	"zlib_upload_window_size" on page 77

- **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...
buffer_size=*bytes*	"buffer_size" on page 40
certificate_company=*company_name*	"certificate_company" on page 41
certificate_name=*name*	"certificate_name" on page 43
certificate_unit=*company_unit*	"certificate_unit" on page 45
client_port=*nnnnn*[-*mmmmm*]	"client_port" on page 46
compression={**zlib**|**none**}	"compression" on page 47
e2ee_type={**rsa**|**ecc**}	"e2ee_type" on page 49
e2ee_public_key=*file*	"e2ee_public_key" on page 50
fips={**y**|**n**}	"fips" on page 51
host=*hostname*	"host" on page 53
network_leave_open={**off**|**on**}	"network_leave_open" on page 59

TLS protocol option	**For more information, see...**
network_name=*name*	"network_name" on page 60
port=*portnumber*	"port" on page 63
timeout=*seconds*	"timeout" on page 67
tls_type={**rsa**|**ecc**}	"tls_type" on page 69
trusted_certificates=*filename*	"trusted_certificates" on page 71
zlib_download_window_size=*window-bits*	"zlib_download_window_size" on page 76
zlib_upload_window_size=*window-bits*	"zlib_upload_window_size" on page 77

- **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=*number*	"buffer_size" on page 40
client_port=*nnnnn*[-*mmmmm*]	"client_port" on page 46
compression={**zlib**|**none**}	"compression" on page 47
custom_header=*header*	"custom_header" on page 48
e2ee_type={**rsa**|**ecc**}	"e2ee_type" on page 49
e2ee_public_key=*file*	"e2ee_public_key" on page 50
http_password=*password*	"http_password" on page 54
http_proxy_password=*password*	"http_proxy_password" on page 55
http_proxy_userid=*userid*	"http_proxy_userid" on page 56
http_userid=*userid*	"http_userid" on page 57
host=*hostname*	"host" on page 53
network_leave_open={**off**|**on**}	"network_leave_open" on page 59
network_name=*name*	"network_name" on page 60
persistent={**off**|**on**}	"persistent" on page 62

HTTP protocol option	For more information, see...
port=*portnumber*	"port" on page 63
proxy_host=*proxy-hostname-or-ip*	"proxy_host" on page 64
proxy_port=*proxy-portnumber*	"proxy_port" on page 65
set_cookie=*cookie-name***=***cookie-value*	"set_cookie" on page 66
timeout=*seconds*	"timeout" on page 67
url_suffix=*suffix*	"url_suffix" on page 73
version=*HTTP-version-number*	"version" on page 75
zlib_download_window_size=*window-bits*	"zlib_download_window_size" on page 76
zlib_upload_window_size=*window-bits*	"zlib_upload_window_size" on page 77

- **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=*number*	"buffer_size" on page 40
certificate_company=*company_name*	"certificate_company" on page 41
certificate_name=*name*	"certificate_name" on page 43
certificate_unit=*company_unit*	"certificate_unit" on page 45
client_port=*nnnnn*[-*mmmmm*]	"client_port" on page 46
compression={**zlib**|**none**}	"compression" on page 47
custom_header=*header*	"custom_header" on page 48
e2ee_type={**rsa**|**ecc**}	"e2ee_type" on page 49
e2ee_public_key=*file*	"e2ee_public_key" on page 50
fips={**y**|**n**}	"fips" on page 51
host=*hostname*	"host" on page 53
http_password=*password*	"http_password" on page 54

HTTPS protocol option	For more information, see...
http_proxy_password=*password*	"http_proxy_password" on page 55
http_proxy_userid=*userid*	"http_proxy_userid" on page 56
http_userid=*userid*	"http_userid" on page 57
network_leave_open={**off**|**on**}	"network_leave_open" on page 59
network_name=*name*	"network_name" on page 60
persistent={**off**|**on**}	"persistent" on page 62
port=*portnumber*	"port" on page 63
proxy_host=*proxy-hostname-or-ip*	"proxy_host" on page 64
proxy_port=*proxy-portnumber*	"proxy_port" on page 65
set_cookie=*cookie-name=cookie-value*	"set_cookie" on page 66
timeout=*seconds*	"timeout" on page 67
tls_type={**rsa**|**ecc**}	"tls_type" on page 69
trusted_certificates=*filename*	"trusted_certificates" on page 71
url_suffix=*suffix*	"url_suffix" on page 73
version=*HTTP-version-number*	"version" on page 75
zlib_download_window_size=*window-size*	"zlib_download_window_size" on page 76
zlib_upload_window_size=*window-bits*	"zlib_upload_window_size" on page 77

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 11 - 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*

Protocols

- TCPIP, TLS, HTTP, HTTPS

Support notes

- Cannot be used in the ML directive of the Redirector.

Default

- Palm - 4K
- CE - 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 187.

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_parms = 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 11 - Introduction*].

Syntax

certificate_company=*organization*

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 187.

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 option" [*MobiLink - Server Administration*]
- "trusted_certificates" on page 71
- "certificate_name" on page 43
- "certificate_unit" on page 45

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_synch_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 11 - Introduction*].

Syntax

certificate_name=*common-name*

Protocols

- TLS, HTTPS

Default

None

Remarks

For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 187.

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 option" [*MobiLink - Server Administration*]
- "trusted_certificates" on page 71
- "certificate_company" on page 41
- "certificate_unit" on page 45

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:

```
mlsrv11
   -c "dsn=SQL Anywhere 11 Demo;uid=DBA;pwd=sql"
   -x https(
     port=9999;
     identity=c:\sa10\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:\sa10\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=\sa10\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 11 - Introduction*].

Syntax

certificate_unit=*organization-unit*

Protocols

- TLS, HTTPS

Default

None

Remarks

For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 187.

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 option" [*MobiLink - Server Administration*]
- "trusted_certificates" on page 71
- "certificate_company" on page 41
- "certificate_name" on page 43

Example

For examples of security, see "certificate_name" on page 43 and "trusted_certificates" on page 71.

client_port

Specify a range of client ports for communication.

Syntax

client_port=*nnnnn*[-*mmmmm*]

Protocols

- TCPIP, TLS, HTTP, HTTPS

Support notes

- Cannot be used in the ML directive of the Redirector.

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 187.

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** }

Protocols

- TCPIP, TLS, HTTP, HTTPS

Support notes

- Not supported on Palm OS or Symbian.
- Cannot be used in the ML directive of the Redirector.

Default

For UltraLite, compression is off by default.

For dbmlsync, zlib compression is used by default.

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, *mlczlib10.dll* must be deployed and for C++ only, applications must call `ULEnableZlibSyncCompression( sqlca )`.

See also

- "zlib_download_window_size" on page 76
- "zlib_upload_window_size" on page 77

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*.

Protocols

- HTTP, HTTPS

Support notes

- Cannot be used in the ML directive of the Redirector.

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 187.

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** }

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 50
- “-x 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*

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 the with TLS/HTTPS protocol option fips. This option is not supported when using ECC. See "fips" on page 51.

See also

- "e2ee_type" on page 49
- "-x 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 11 - Introduction*].

Syntax

fips={ **y** | **n** }

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 49 and "e2ee_public_key" on page 50.

For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 187.

See also

- "tls_type" on page 69
- "e2ee_type" on page 49

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:

```
mlsrv11
  -c "dsn=SQL Anywhere 11 Demo;uid=DBA;pwd=sql"
  -x tls(
    port=9999;
    tls_type=rsa;
    fips=y;
    identity=c:\sa10\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*

Protocols

- TCPIP, TLS, HTTP, HTTPS

Default

- Windows Mobile - the default value is the IP address of the desktop computer the device has an 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 ActiveSync partnership.

For the Palm Computing Platform, the default value of localhost refers to the device. You should supply an explicit host name or IP address to connect to a desktop computer.

For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 187.

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_password

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

Syntax

http_password=*password*

Protocols

- HTTP, HTTPS

Support notes

- Cannot be used in the ML directive of the Redirector.

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 dbmlsync, see "CommunicationAddress (adr) extended option" on page 187.

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 57
- "http_proxy_password" on page 55
- "http_proxy_userid" on page 56

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*

Protocols

- HTTP, HTTPS

Support notes

- Cannot be used in the ML directive of the Redirector.

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 187.

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 54
- "http_userid" on page 57
- "http_proxy_userid" on page 56

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*

Protocols

- HTTP, HTTPS

Support notes

- Cannot be used in the ML directive of the Redirector.

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 187.

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 54
- "http_userid" on page 57
- "http_proxy_password" on page 55

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*

Protocols

- HTTP, HTTPS

Support notes

- Cannot be used in the ML directive of the Redirector.

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 187.

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 54
- “http_proxy_password” on page 55
- “http_proxy_userid” on page 56

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_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 11 - Introduction*].

Syntax

identity_name=*name*

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 11 - Introduction*].

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 }

Protocols

- TCPIP, TLS, HTTP, HTTPS

Support notes

- Cannot be used in the ML directive of the Redirector.

Default

On devices other than Palm, the default is **off**.

On the Palm, the default is **on**.

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 187.

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 60

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*

Protocols

- TCPIP, TLS, HTTP, HTTPS

Support notes

- Cannot be used in the ML directive of the Redirector.

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 187.

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 121
- "network_leave_open" on page 59

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** }

Protocols

- HTTP, HTTPS

Support notes

- Cannot be used in the ML directive of the Redirector.

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.

Except on Palm devices, 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 or redirector, a persistent connection may cause problems.

For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 187.

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*

Protocols

- TCPIP, 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 187.

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*

Protocols

- HTTP, HTTPS

Support notes

- Cannot be used in the ML directive of the Redirector.

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 187.

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*

Protocols

- HTTP, HTTPS

Support notes

- Cannot be used in the ML directive of the Redirector.

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 187.

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*,...

Protocols

- HTTP, HTTPS

Support notes

- Cannot be used in the ML directive of the Redirector.

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 dbmlsync, see "CommunicationAddress (adr) extended option" on page 187.

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*

Protocols

- TCPIP, TLS, HTTP, HTTPS

Default

240 seconds

Support notes

- Cannot be used in the ML directive of the Redirector.

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, it is possible that 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 187.

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 11 - Introduction*].

Syntax

tls_type=*cipher*

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 187.

See also

- "Configuring MobiLink clients to use transport-layer security" [*SQL Anywhere Server - Database Administration*]
- "fips" on page 51
- "Encrypting MobiLink client/server communications" [*SQL Anywhere Server - Database Administration*]
- "-x option" [*MobiLink - Server Administration*]
- "certificate_company" on page 41
- "certificate_name" on page 43
- "certificate_unit" on page 45
- "trusted_certificates" on page 71

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:

```
mlsrv11
  -c "dsn=SQL Anywhere 11 Demo;uid=DBA;pwd=sql"
  -x tls(
    port=9999;
    tls_type=rsa;
    identity=c:\sa10\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 11 - Introduction*].

Syntax

trusted_certificates=*filename*

Syntax 2 (Palm OS)

trusted_certificates=vfs:[*volume-label***:**| *volume-ordinal***:**]*filename*

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, ulcreate, 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 32-bit 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.

Trusted certificates are supported for the Palm OS file system (not record-based data stores). On Palm OS, *volume-label* can be **INTERNAL** for the built-in drive, **CARD** for the expansion card, or the label name of the volume. Alternatively, you can use *volume-ordinal* to identify the volume (the default is 0, which is the first volume enumerated by the platform). The *filename* must be the full path to the file, following the filename and path naming conventions of the Palm platform.

For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 187.

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 option" [*MobiLink - Server Administration*]
- "tls_type" on page 69
- "certificate_company" on page 41
- "certificate_name" on page 43
- "certificate_unit" on page 45

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:

```
mlsrv11
   -c "dsn=SQL Anywhere 11 Demo;uid=DBA;pwd=sql"
   -x https(
     port=9999;
     identity=c:\sa10\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:\sa10\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");
```

On an UltraLite client running Palm OS, the stream and stream_parms can be set like this:

```
info.stream = "https";
info.stream_parms = "trusted_certificates=vfs:/rsaroot.crt;port=9999";
```

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 the type of Redirector you are using:

Redirector	Syntax of *suffix*
ISAPI	`exe_dir/iaredirect.dll/ml/[server-group/]` where *exe-dir* is the location of *iaredirect.dll*.
NSAPI	`mlredirect/ml/[server-group/]` where *mlredirect* is a name mapped in your *obj.conf* file.
Apache	whatever you chose in the Redirector's <location> tag in the *httpd.conf* file.
Servlet	`iaredirect/ml/`
M-Business Anywhere	whatever you chose in the Redirector's <location> tag in the *sync.conf* file.

Protocols

- HTTP, HTTPS

Support notes

- Cannot be used in the ML directive of the Redirector, but is set on the client for the Redirector.

Default

The default is **MobiLink**.

Remarks

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

Only some Redirectors support server groups. For details, see http://www.sybase.com/detail?id=1061837.

For information about how to set this option when using the Redirector, see "Configuring MobiLink clients and servers for the Redirector" [*MobiLink - Server Administration*].

For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 187.

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

- "Configuring MobiLink clients and servers for the Redirector" [*MobiLink - Server Administration*]
- "MobiLink server groups" [*MobiLink - Server Administration*]

Example

The following SQL statement creates a synchronization user called sales5322 that synchronizes over HTTPS. Assume that the MobiLink server runs behind the corporate firewall, and synchronization requests are redirected to it using the Redirector (a reverse proxy to an NSAPI web server). The MobiLink user synchronizes to the URL *https://www.mycompany.com:80/mlredirect/ml/*.

```
CREATE SYNCHRONIZATION USER sales5322
TYPE https
ADDRESS 'host=www.mycompany.com;port=80;url_suffix=mlredirect/ml/'
```

Example

The following example sets up HTTP through the ISAPI Redirector for an UltraLite client. Command lines must be entered all on one line.

```
mlsrv11 -c "dsn=SQL Anywhere 11 CustDB"
  -dl -fr -ot mlserver.mls -zu+ -v+
  -x http(port=8081)
```

To synchronize with ulsync, run a command such as the following:

```
ulsync -c "dbf=custdb.udb"
  -v "MobiLinkUid=50;ScriptVersion=custdb 11.0;
   Stream=http(port=80;url_suffix=Scripts/iaredirect.dll/ml/)"
```

version

Specify the version of HTTP to use for synchronization.

Syntax

version=*HTTP-version-number*

Protocols

- HTTP, HTTPS

Support notes

- Cannot be used in the ML directive of the Redirector.

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 187.

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*

Protocols

- TCPIP, TLS, HTTP, HTTPS

Support notes

- Cannot be used in the ML directive of the Redirector.
- Not supported on Palm OS or Symbian.

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 *mlczlib10.dll* must be deployed.

For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 187.

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 47
- "zlib_upload_window_size" on page 77

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*

Protocols

- TCPIP, TLS, HTTP, HTTPS

Support notes

- Cannot be used in the ML directive of the Redirector.
- Not supported on Palm OS or Symbian.

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

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

To support zlib compression in UltraLite, an application must call `ULEnableZlibSyncCompression( sqlca )` and *mlczlib10.dll* must be deployed.

For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 187.

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 47
- "zlib_download_window_size" on page 76

Example

The following option sets compression for download only:

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

Schema changes in remote clients

Contents

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.

Caution
Perform a successful synchronization just before changing schema. There should be no open transactions when you upgrade the schema.

Schema upgrades for SQL Anywhere remote databases

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

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_dbmlsync_schema_upgrade hook to upgrade your schema.

See "sp_hook_dbmlsync_schema_upgrade" on page 290.

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_dbmlsync_schema_upgrade hook. See "sp_hook_dbmlsync_schema_upgrade" on page 290.

   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_dbmlsync_schema_upgrade hook. See "sp_hook_dbmlsync_schema_upgrade" on page 290.
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_dbmlsync_schema_upgrade hook. See "sp_hook_dbmlsync_schema_upgrade" on page 290.
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 passthrough

Contents

Introduction to SQL passthrough

The SQL Passthrough feature allows you to download scripts of SQL statements from a consolidated database to a SQL Anywhere or UltraLite client, and have those SQL statements executed on the client at an appropriate time. The scripts are numbered and are guaranteed to be executed on the client in order.

After a script is executed or an attempt is made to execute it, status is sent back to the consolidated database with the next synchronization so that both successful runs and failures can be centrally monitored. When an error occurs executing a script on a client, no further scripts are executed on that client until the status has been uploaded to the consolidated database and the consolidated has downloaded instructions to the client on how to proceed. These instructions may mean retrying the existing script, skipping it, or downloading a new script to fix the error.

Below is an overview of the steps required to use SQL Passthrough:

- Create a script and store it in the consolidated database. See "Creating a script" on page 86 and "ml_add_passthrough_script system procedure" [*MobiLink - Server Administration*].
- Create one or more passthrough entries for the script which identify the clients that should execute it. See "Creating passthrough entries" on page 86 and "ml_add_passthrough system procedure" [*MobiLink - Server Administration*].
- Download the script. See "Downloading scripts" on page 87 and "ml_add_passthrough system procedure" [*MobiLink - Server Administration*].
- Execute the script. See "Executing scripts" on page 87.
- Capture the results. See "Capturing script results" on page 90.
- Review the results. See "Reviewing script results" on page 90.
- Handle errors, if necessary. See "Handling script errors" on page 91.

Creating a script

Scripts are created and stored on the consolidated database using the ml_add_passthrough_script stored procedure. See "ml_add_passthrough_script system procedure" [*MobiLink - Server Administration*].

Creating passthrough entries

The ml_add_passthrough system procedure is used to specify which MobiLink clients should receive the passthrough script that was created on the consolidated database. See "ml_add_passthrough system procedure" [*MobiLink - Server Administration*].

Downloading scripts

Scripts are downloaded from the consolidated database to the MobiLink client automatically during synchronization. Scripts are **not** downloaded in the following situations:

- When a file-based download is performed.
- When a ping is performed.
- When a download is restarted.

SQL passthrough script storage on SQL Anywhere clients

SQL passthrough scripts downloaded from the consolidated database to a SQL Anywhere client are stored in the dbo.sync_passthrough_script table.

SQL passthrough script storage on UltraLite clients

SQL passthrough scripts downloaded from the consolidated database to an UltraLite client are stored in the syssql table.

Executing scripts

Once a script is stored on a MobiLink client database, it can be executed either automatically or manually. Regardless of how scripts are executed, they must be executed in order by run_order.

For SQL Anywhere clients, scripts are always executed with DBA authority under the DBO account.

Executing a script manually on a SQL Anywhere client

Any script may be executed manually. For SQL Anywhere clients, scripts are executed manually using the sync_get_next_passthrough_script and sync_execute_next_passthrough_script functions.

The sync_get_next_passthrough_script function takes no parameters and returns the run_order of the next script to be executed. You can then find out about that script by querying the dbo.sync_passthrough_script table.

dbo.sync_passthrough_script table

The dbo.sync_passthrough_script table is defined as follows:

Column name	Description
run_order	INTEGER. The run_order parameter determines the order in which scripts are applied on the remote database. Scripts are always applied in order by run_order. This value must be a non-negative integer.
script_id	INTEGER. This value uniquely identifies the script.

Column name	Description
script_name	VARCHAR(128). The name of the script. This column corresponds to the script_name value specified for the script when ml_add_passthrough_script was called on the consolidated database.
flags	BIGINT. The flags column contains the information specified in the flags parameter passed to the ml_add_passthrough_script stored procedure. The flags specified are encoded into an integer by converting each specified flag into the value shown below, and combining the resultant values together with OR operators. ● **manual** Indicates that the script may only be run in manual execution mode. By default, all scripts can be run in either automatic or manual execution modes. ● **exclusive** Indicates that the script may only be automatically executed at the end of a synchronization where exclusive locks were obtained on all tables being synchronized. This option is ignored if the affected_ publications value lists no publications. This option is only meaningful to SQL Anywhere remotes. ● **schema_diff** Indicates that the script should be run in schema-diffing mode. In this mode, the database schema is altered to match the schema described in the script. For example, a create statement for an existing table is treated as an alter statement. This flag only applies to scripts run on UltraLite remotes.
affected_pubs	LONG VARCHAR. A list of publications that must be synchronized before the script is run. This column corresponds to the affected_pubs value specified for the script when ml_add_passthrough_script was called.
script	LONG VARCHAR. The contents of the passthrough script. This column corresponds to the script value specified for the script when ml_add_passthrough_script was called.
description	VARCHAR(2000). A comment or description of the script. This column corresponds to the description value specified for the script when ml_add_passthrough_script was called.

The sync_get_next_passthrough_script function returns null if there are no more scripts to execute or if the last script executed generated an error and no instructions on how to proceed have yet be received from the server.

The sync_execute_next_passthrough_script function takes no parameters. It executes the next script and updates progress and status information in the database so that the results of the script can be uploaded to the consolidated database later. No script is executed if the last script returned an error and no instructions have yet been received from the MobiLink server on how to handle the error. If a script is executed, the run order of that script is returned. If no script is executed, null is returned. See "Capturing script results" on page 90.

Executing a script manually on an UltraLite client

For UltraLite clients, several ESQL API methods are available to manually apply scripts. The methods are:

- "GetSQLPassthroughScriptCount method" [*UltraLite - .NET Programming*]
- "ExecuteSQLPassthroughScripts method" [*UltraLite - .NET Programming*]
- "ExecuteNextSQLPassthroughScript method" [*UltraLite - .NET Programming*]

Equivalent methods are available for the C++, UltraLite.NET, and M-Business Anywhere interfaces.

Executing a script automatically on a SQL Anywhere client

On SQL Anywhere clients, an attempt is made to execute any waiting scripts at the end of each synchronization. The available scripts are ordered by run_order and executed one at a time until one of the following occurs:

- All the scripts have been executed.
- A script fails.
- A script is reached that cannot be executed automatically.

A script cannot be executed automatically if any of the following are true:

- The **manual** flag was specified when the script was created.
- The script has a non-empty **affected publications** value plus one or more of the following conditions:
  - No upload was performed.
  - The upload failed.
  - One or more publications listed in the **affected publications** value were not synchronized.
  - The **exclusive** flag was specified when the script was created and exclusive locks were not obtained on all synchronizing tables when synchronization began.

Note
Download-only publications should never be listed as affected publications.

At the beginning of a synchronization, dbmlsync may choose to obtain locks on the synchronizing tables that are more restrictive than those requested using the LockTables extended option to ensure that scripts can be executed at the end of the synchronization. For example, if LockTables is set to SHARE but the next script available for execution requires exclusive locks, then exclusive locks may be obtained.

Executing a script automatically on an UltraLite client

On UltraLite clients, scripts that are not marked as **manual** are run automatically the next time that the database is started, unless the connection parameter **dont_run_scripts** has been set.

Script execution progress can be provided during both manual and automatic execution if a progress observer callback has been provided, which is defined as follows:

```
typedef void(UL_CALLBACK_FN *ul_sql_passthrough_observer_fn)
    ( ul_sql_passthrough_status * status );
```

The ul_sql_passthrough_status structure is defined as follows:

```
typedef struct {
    ul_sql_passthrough_state    state; // current state
    ul_u_long  script_count; // total number of scripts to execute
    ul_u_long  cur_script; // current script being executed (1-based)
    ul_bool  stop;  // set to true to stop script execution
         // can only be set in the starting state
    ul_void *  user_data; // user data provided in register call
    SQLCA *  sqlca;
} ul_sql_passthrough_status;
```

The progress observer callback is registered via the following method:

```
UL_FN_SPEC ul_ret_void UL_FN_MOD ULRegisterSQLPassthroughCallback(
SQLCA *                               sqlca,
ul_sql_passthrough_observer_fn        callback,
ul_void *                             user_data );
```

Capturing script results

Regardless of whether a script is executed manually or automatically, the results of the script execution are stored on the MobiLink client.

On SQL Anywhere clients, the results are stored in the dbo.sync_passthrough_status table. The results consist of the time at the remote database when the script was executed and an indication of whether the script succeeded or reported an error. In addition, if an error was reported the SQL code and the text of the error message is stored.

On UltraLite clients the results are stored in the syssql table. The results consist of the time at the remote database when the script was executed and an indication of whether the script succeeded or reported an error. In addition, if an error was reported the SQL code, the line number in the script of the statement that failed, and a list of error parameters is stored.

Reviewing script results

Both UltraLite and SQL Anywhere clients upload the results of scripts they have executed as part of each synchronization. The MobiLink server stores the uploaded results in the ml_passthrough_status table in the consolidated database. By examining this table you can determine the success of the passthrough scripts that were distributed to the clients. See "ml_passthrough_status" [*MobiLink - Server Administration*].

Handling script errors

Caution
The SQL passthrough feature is extremely powerful and must be used with **caution**. It is particularly important that scripts be well tested because SQL passthrough script errors can potentially disable or damage all of your remotes. Avoid errors with thorough testing.

When a SQL passthrough script on the client generates errors, the errors must be resolved at the consolidated database or the client is not able to run any more SQL passthrough scripts.

The -vo server option for mlsrv11 can be used to capture SQL passthrough activity on the MobiLink server that can help you resolve errors. See "-v option" [*MobiLink - Server Administration*].

The primary mechanism for resolving an error on a client is to add a row to the ml_passthrough_repair table using the ml_add_passthrough_repair system procedure. Rows in the ml_passthrough_repair table describe the action the client should take when a specific script generates a specific error code.

See also

- "ml_passthrough_repair" [*MobiLink - Server Administration*]
- "ml_add_passthrough_repair system procedure" [*MobiLink - Server Administration*]
- "ml_delete_passthrough_repair system procedure" [*MobiLink - Server Administration*]

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

Contents

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, register the user with the consolidated database, and subscribe the MobiLink user to the publication.

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.
2. Create one or more publications in the remote database.

   See "Publishing data" on page 100.
3. Create MobiLink users in the remote database.

   See "Creating MobiLink users" on page 107.
4. Register users with the consolidated database.

   See "Adding MobiLink user names to the consolidated database" on page 11.
5. Subscribe MobiLink users to one or more of the publications.

   See "Creating synchronization subscriptions" on page 110.

Deploying remote databases

To deploy SQL Anywhere remote databases, you need to create the databases and add the appropriate publications and subscriptions. 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 pre-populate 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 97.

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 models" [*MobiLink - Getting Started*].

See also

- "Deploying SQL Anywhere MobiLink clients" [*MobiLink - Server Administration*]
- "First synchronization always works" on page 99

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'
        OPTION MEM=''
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 plug-in to Sybase Central.

The remote ID must be unique within your synchronization system.

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 14

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 by setting the progress column of the ml_user table to 0 (zero) for this user.

For more information about the ml_user MobiLink system table, see "ml_user" [*MobiLink - Server Administration*].

For more information about upgrading, see "Upgrading SQL Anywhere MobiLink clients" [*SQL Anywhere 11 - 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. In most cases, 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 option" on page 170.

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 dbmlsync 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 dbmlsync -n option with two publications (-n pub1,pub2), and pub1 has synchronized before but pub2 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*]
- "ml_subscription" [*MobiLink - Server Administration*]
- "Transaction log files" on page 114

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 Admin mode)

1. Connect to the remote database as a user with DBA authority, using the SQL Anywhere 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 customer table:

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

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 customer,
 TABLE sales_order,
 TABLE sales_order_items,
 TABLE product
)
```

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 any user that subscribes to both publications is unable to synchronize.
- An article must include all the primary key columns in the table.

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

1. Connect to the remote database as a user with DBA authority using the SQL Anywhere 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 customer table:

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

Publishing only some rows in a table

When no WHERE clause is specified in a publication definition, all changed rows in the publication 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 Admin mode)

1. Connect to the remote database as a user with DBA authority using the SQL Anywhere 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 SalesOrder 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 79.

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

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 Admin mode)

1. Connect to the remote database as a user who owns the publication or as a user with DBA authority using the SQL Anywhere 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 Admin mode)

1. Connect to the database as a user who owns the publication or as a user with DBA authority using the SQL Anywhere 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 customer table to the pub_contact publication.

  ```
  ALTER PUBLICATION pub_contact (
   ADD TABLE customer
  )
  ```

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. Before dropping the publication, you must drop all subscriptions connected to it.

You must have DBA authority to drop a publication.

To delete a publication (Sybase Central Admin mode)

1. Connect to the remote database as a user with DBA authority using the SQL Anywhere 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 Admin mode)

1. Connect to the database from the SQL Anywhere plug-in as a user with DBA authority.
2. Click the **MobiLink Users** folder.
3. Choose **File** » **New** » **User**.
4. In the **What Do You Want To Name The New 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 107.

For information about registering MobiLink users, see "Adding MobiLink user names to the consolidated database" on page 11.

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 183.

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

1. Connect to the database from the SQL Anywhere 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 292.

See also

- "Using dbmlsync extended options" on page 113

Dropping MobiLink users

You must drop all subscriptions for a MobiLink user before you drop the user from a remote database.

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

1. Connect to the database from the SQL Anywhere 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 100. For information about creating MobiLink users, see “Creating MobiLink users” on page 107.

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 customer and sales_order tables in the SQL Anywhere sample database, you could use the following statements.

1. First, publish the customer and sales_order tables. Give the publication the name testpub.

   ```
   CREATE PUBLICATION testpub
    (TABLE customer, TABLE sales_order)
   ```

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 subscription that links the user and the publication.

   ```
   CREATE SYNCHRONIZATION SUBSCRIPTION TO testpub
    FOR demo_ml_user
    TYPE tcpip
    ADDRESS 'host=localhost;port=2439;'
    OPTION sv='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 Admin mode)

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 Admin mode)

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 of MobiLink user jsmith to a publication named pub_orders.

```
DROP SYNCHRONIZATION SUBSCRIPTION
FOR jsmith TO pub_orders
```

See "DROP SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*].

Initiating synchronization

The client always initiates MobiLink synchronization. In the case of a SQL Anywhere client, synchronization is initiated by running the dbmlsync utility. This utility connects to and synchronizes a SQL Anywhere remote database.

See "MobiLink SQL Anywhere client utility (dbmlsync)" on page 129.

You can specify connection parameters on the dbmlsync command line using the -c option. These parameters are for the remote database. If you do not specify connection parameters, a connection window appears, asking you to supply the missing connection parameters and startup options.

See "-c option" on page 142.

Client network protocol options can be stored in the synchronization subscription, publication or user in the remote database; or can be specified on the dbmlsync command line. These are used to locate the appropriate MobiLink server.

See "CommunicationAddress (adr) extended option" on page 187.

Permissions for dbmlsync

When dbmlsync connects to a database, it must have permissions to apply all the changes being made. The dbmlsync command line contains the password for this connection. This could present a security issue.

To avoid security problems, grant a user (other than DBA) REMOTE DBA authority, and use this user ID in the dbmlsync connection string. A user ID with REMOTE DBA authority has DBA authority only when the connection is made from the dbmlsync utility. Any other connection using the same user ID is granted no special authority.

See "GRANT REMOTE DBA statement [MobiLink] [SQL Remote]" [*SQL Anywhere Server - SQL Reference*].

Customizing synchronization

See "Customizing dbmlsync synchronization" on page 123.

Using dbmlsync 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 dbmlsync command line.

For a complete list of extended options, see "MobiLink SQL Anywhere client extended options" on page 183.

To override an extended option on the dbmlsync command line

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

  ```
  dbmlsync -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 can 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 189
- "CommunicationAddress (adr) extended option" on page 187

Transaction log files

In most cases, dbmlsync determines what to upload by using the SQL Anywhere transaction log. The offset indicates the point to which all operations for a subscription have been uploaded and acknowledged.

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.

The transaction log may not be required if you implement scripted upload 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 98
- “The transaction log” [*SQL Anywhere Server - Database Administration*]
- “Initialization utility (dbinit)” [*SQL Anywhere Server - Database Administration*]
- “Scripted upload” on page 373

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.

```
dbmlsync -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, dbmlsync must ensure that no rows in the download are modified between the time the upload is built and the time the download is applied.

On all platforms except Windows Mobile, by default, dbmlsync obtains a shared lock on all tables mentioned in any publication being synchronized. On Windows Mobile, by default, dbmlsync obtains an exclusive lock. Dbmlsync obtains the lock before it begins building the upload, and it maintains the lock until the download is applied.

For more information about locks, see “Row locks” [*SQL Anywhere Server - SQL Usage*].

The following options let you customize this locking behavior:

- -d option
- LockTables option

-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 dbmlsync -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 option” on page 143.

LockTables option

An alternative way to protect data integrity is to set the extended option LockTables to OFF, which prevents an article's tables from being locked. This causes dbmlsync to track all rows that are modified after the upload has been built. When the download is received, it is not applied if any rows in the download have been modified. Dbmlsync then retries the synchronization. The retry succeeds unless a new download conflict is detected.

For more information, see “LockTables (lt) extended option” on page 203.

If a conflict is detected, the download phase is canceled and the download operations rolled back to avoid overwriting the new change. The dbmlsync 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, dbmlsync retries synchronization until success is achieved. You can limit the number of retries using the extended option ConflictRetries. Setting ConflictRetries to -1 causes dbmlsync to retry until success is achieved. Setting it to a non-negative integer causes dbmlsync to retry for not more than the specified number of times.

For more information, see "ConflictRetries (cr) extended option" on page 190.

Initiating synchronization from an application

You may want to include the features of dbmlsync in your application, rather than provide a separate executable to your remote users.

There are two ways to do this:

- Use the Dbmlsync Integration Component.

  For more information, see "Dbmlsync integration component (deprecated)" on page 337.

- If you are developing in any language that can call a DLL, then you can access dbmlsync 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 11 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.

  For more information, see:

  - "DBTools interface for dbmlsync" on page 365
  - "DBSynchronizeLog function" [*SQL Anywhere Server - Programming*]
  - "a_sync_db structure" [*SQL Anywhere Server - Programming*]

Using ActiveSync synchronization

ActiveSync is synchronization software for Microsoft Windows Mobile handheld devices. ActiveSync governs synchronization between a Windows Mobile device and a desktop computer. A MobiLink provider for ActiveSync governs synchronization to the MobiLink server.

Setting up ActiveSync synchronization for SQL Anywhere clients involves the following steps:

- Configure the SQL Anywhere remote database for ActiveSync synchronization.

  See "Configuring SQL Anywhere remote databases for ActiveSync" on page 117.

- Install the MobiLink provider for ActiveSync.

  See "Installing the MobiLink provider for ActiveSync" on page 118.

- Register the SQL Anywhere client for use with ActiveSync.

  See "Registering SQL Anywhere clients for ActiveSync" on page 119.

If you use ActiveSync synchronization, synchronization must be initiated from the ActiveSync software. The MobiLink provider for 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 ActiveSync cannot invoke synchronization from this state.

For information about scheduling synchronization, see "Scheduling synchronization" on page 121.

Configuring SQL Anywhere remote databases for ActiveSync

To configure your SQL Anywhere remote database for 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 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 ActiveSync, the communication takes place in two stages: from the dbmlsync utility on the device to the MobiLink provider for ActiveSync on the desktop computer, and from the desktop computer to the MobiLink server. The ADDRESS clause specifies the communication between MobiLink provider for 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 ActiveSync

Before you register your SQL Anywhere MobiLink client for use with ActiveSync, you must install the MobiLink provider for ActiveSync using the installation utility (*mlasinst.exe*).

The SQL Anywhere for Windows Mobile installer installs the MobiLink provider for 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 ActiveSync you must register each application separately. For instructions, see "Registering SQL Anywhere clients for ActiveSync" on page 119.

To install the MobiLink provider for ActiveSync

1. Ensure that you have the 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 "ActiveSync provider installation utility (mlasinst)" on page 27.
3. Restart your computer.

   ActiveSync does not recognize new providers until the computer is restarted.
4. Enable the MobiLink provider.

   For Windows versions prior to Vista:

   - In the 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 ActiveSync" on page 119.

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 ActiveSync

You can register your application for use with ActiveSync either by using the ActiveSync provider install utility or using the ActiveSync software itself. This section describes how to use the ActiveSync software.

For information about the alternative approach, see "ActiveSync provider installation utility (mlasinst)" on page 27.

To register the SQL Anywhere client for use with ActiveSync

1. Ensure that the MobiLink provider for ActiveSync is installed.

   For information, see "Installing the MobiLink provider for ActiveSync" on page 118.

2. Start the ActiveSync software on your desktop computer.

3. For Windows prior to Vista:

   - From the **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 ActiveSync user interface.
   - **Class name** The class name for the dbmlsync client, as set using the -wc option.

  For more information, see “-wc option” on page 181.

  - **Path**   The location of the dbmlsync application on the device.
  - **Arguments**   Any command line arguments to be used when ActiveSync starts dbmlsync.

    You start dbmlsync in one of two modes:

    - If you specify scheduling options, dbmlsync enters hover mode. In this case, use the dbmlsync -wc option with a matching value in the class name setting.

      For more information, see “-wc option” on page 181 and “Scheduling synchronization” on page 121.

    - Otherwise, dbmlsync is not in hovering mode. In this case, use -k to shut down dbmlsync.

      For more information, see “-k option (deprecated)” on page 155.

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

Scheduling synchronization

You can set up dbmlsync to synchronize periodically based on rules you define. There are two ways you can set this up:

- Use the dbmlsync extended option SCHEDULE to initiate synchronization at specific times of the day or week or at regular intervals. In this case, dbmlsync remains running until stopped by the user.

  See "Setting up scheduling with dbmlsync options" on page 121.

- Use dbmlsync 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 dbmlsync programmatically from your hook code.

  See "Initiating synchronization with event hooks" on page 122.

Hovering

When scheduling options or hooks are specified, dbmlsync goes into hovering mode. Hovering is a feature that reduces the amount of time spent scanning the log. You can improve the performance benefits of hovering by setting the dbmlsync extended option HoverRescanThreshold or by using the dbmlsync stored procedure sp_hook_dbmlsync_log_rescan.

For more information, see:

- "HoverRescanThreshold (hrt) extended option" on page 199
- "sp_hook_dbmlsync_log_rescan" on page 276

Setting up scheduling with dbmlsync options

Instead of running dbmlsync in a batch fashion, where it synchronizes and then shuts down, you can set up a SQL Anywhere client so that dbmlsync runs continuously, synchronizing at predetermined times.

You specify the synchronization schedule as an extended option. It can be specified either on the dbmlsync command line or it can be stored in the database for the synchronization user, subscription, or publication.

For more information about scheduling syntax, see "Schedule (sch) extended option" on page 212.

For more information about extended options, see:

- "MobiLink SQL Anywhere client extended options" on page 183
- "-eu option" on page 153

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 dbmlsync -is option. The -is option instructs dbmlsync to ignore scheduling that is specified with the scheduling extended option. For more information, see “-is option” on page 154.

To add scheduling from the dbmlsync command line

- Set the schedule extended option. Extended options are set with -e or -eu. For example,

  ```
  dbmlsync -e "sch=weekday@11:30am-12:30pm" ...
  ```

If scheduled synchronization is specified in either place, dbmlsync does not shut down after synchronizing, but runs continuously.

Initiating synchronization with event hooks

There are dbmlsync 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 273.

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 253.

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 235.

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 "Introduction to the Dbmlsync API" on page 308.

- **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 365.

Scripted upload

You can also override the use of the client transaction log and define your own upload stream. See "Scripted upload" on page 373.

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.log*.

```
dbmlsync -o dbmlsync.log ...
```

Logging dbmlsync activity is particularly useful during the development process and when troubleshooting. Verbose output is not recommended for normal operation in a production environment because it can slow performance.

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 option" on page 160
- "-ot option" on page 162
- "-os option" on page 161

You can control what information is logged to the message log file and displayed in the dbmlsync messages window using the -v option.

For more information, see "-v option" on page 180.

You can manage log files using the delete_old_logs option.

For more information, see "delete_old_logs option [MobiLink client] [SQL Remote] [Replication Agent]" [*SQL Anywhere Server - Database Administration*].

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.

SQL Anywhere transaction log

See "Transaction log files" on page 114.

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/SQLAnywhere11*.

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 "Setting environment variables on Unix and Mac OS X" [*SQL Anywhere Server - Database Administration*].

      If *dsn-name* has spaces, surround the string with double quotes. For example:

      ```
      dsn="SQL Anywhere 11 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 "mlsrv11 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/SQLAnywhere11*.

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 131.

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 11 Demo"
   -c "uid=DBA;pwd=sql;dbf=/Applications/SQLAnywhere11/System/demo.db"
   ```

3. Run the MobiLink server. For example:

   ```
   mlsrv11 -c "dsn=SQL Anywhere 11 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)

Contents

dbmlsync syntax

Use the dbmlsync utility to synchronize SQL Anywhere remote databases with a consolidated database.

Syntax

dbmlsync [*options*] [*transaction-logs-directory*]

Option	Description
@data	Read in options from the specified environment variable or configuration file. See "@data option" on page 135.
-a	Do not prompt for input again on error. See "-a option" on page 136
-ap	Specify authentication parameters. See "-ap option" on page 137.
-ba *filename*	Apply a download file. See "-ba option" on page 138.
-bc *filename*	Create a download file. See "-bc option" on page 139.
-be *string*	When creating a download file, add a string. See "-be option" on page 140.
-bg	When creating a download file, make it suitable for new remotes. See "-bg option" on page 141.
-c *connection-string*	Supply database connection parameters in the form *parm1=value1*; *parm2=value2*,... If you do not supply this option, a window appears and you must supply connection information. See "-c option" on page 142.
-d	Drop any other connections to the database whose locks conflict with the articles to be synchronized. See "-d option" on page 143
-dc	Continue a previously failed download. See "-dc option" on page 144.
-dl	Display log messages on the dbmlsync messages window. See "-dl option" on page 145.
-do	Disables scanning of offline transaction logs. See "-do option" on page 146.
-drs *bytes*	For restartable downloads, specify the maximum amount of data that may need to be resent after a communications failure. See "-drs option" on page 147.
-ds	Perform a download-only synchronization. See "-ds option" on page 148.

Option	Description
-e "*option=value*"...	Specify extended options. See "MobiLink SQL Anywhere client extended options" on page 183.
-eh	Ignore errors that occur in hook functions.
-ek *key*	Specify encryption key. See "-ek option" on page 151.
-ep	Prompt for encryption key. See "-ep option" on page 152.
-eu	Specify extended options for upload defined by most recent -n option. See "-eu option" on page 153.
-is	Ignore schedule. See "-is option" on page 154.
-k	Close window on completion. See "-k option (deprecated)" on page 155.
-l	List available extended options. See "-l option" on page 156.
-mn *password*	Specify new MobiLink password. See "-mn option" on page 157.
-mp *password*	Specify MobiLink password. See "-mp option" on page 158.
-n *name*	Specify synchronization publication name(s). See "-n option" on page 159.
-o *logfile*	Log output messages to this file. See "-o option" on page 160.
-os *size*	Specify a maximum size for the message log file, at which point the log is renamed. See "-os option" on page 161.
-ot *logfile*	Delete the contents of the message log file and then log output messages to it. See "-ot option" on page 162.
-p	Disable logscan polling. See "-p option" on page 163.
-pc+	Maintain an open connection to the MobiLink server between synchronizations. See "-pc option" on page 164
-pd *dllname*;...	Preload specified DLLs for Windows Mobile. See "-pd option" on page 165.
-pi	Test that you can connect to MobiLink. See "-pi option" on page 166.
-pp *number*	Set logscan polling period. See "-pp option" on page 167.
-q	Run in minimized window. See "-q option" on page 168.

Option	Description
-qc	Shut down dbmlsync when synchronization is finished. See "-qc option" on page 169.
-r[**a** | **b**]	Use client progress values for upload retry. See "-r option" on page 170.
-sc	Reload schema information before each synchronization. See "-sc option" on page 171.
-sp *sync profile*	Add options from the synchronization profile to the synchronization options specified on the command line. See "-sp option" on page 172.
-tu	Perform transactional upload. See "-tu option" on page 173.
-u *ml_username*	Specify the MobiLink user to synchronize. See "-u option" on page 175.
-ui	For Linux with X window, starts dbmlsync in shell mode if a usable display isn't available. See "-ui option" on page 176.
-uo	Perform upload-only synchronization. See "-uo option" on page 177.
-urc *row-estimate*	Specify an estimate of the number of rows to upload. See "-urc option" on page 178.
-ux	For Solaris and Linux, open the dbmlsync messages window. See "-ux option" on page 179.
-v[*levels*]	Verbose operation. See "-v option" on page 180.
-wc *classname*	Specify a window class name. See "-wc option" on page 181.
-x	Rename and restart the transaction log. See "-x option" on page 182.
transaction-logs-directory	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 transaction log archive files, both 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 deleted and the file has been renamed since you last synchronized
- you run the dbmlsync utility from a directory other than the one where the renamed log files are stored

For more information, see “Transaction log files” on page 114.

dbmlsync event hooks There are also dbmlsync client stored procedures that can help you customize the synchronization process. For more information, see “Introduction to dbmlsync hooks” on page 237 and “Event hooks for SQL Anywhere clients” on page 235.

Using dbmlsync For more information about using dbmlsync, see “Initiating synchronization” on page 113.

See also

- “Initiating synchronization” on page 113
- “Event hooks for SQL Anywhere clients” on page 235
- “Dbmlsync API” on page 307
- “DBTools interface for dbmlsync” on page 365

@data option

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

Syntax

dbmlsync *@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 option

Specifies that dbmlsync should not display a window prompt for input again on error.

Syntax

dbmlsync -a ...

-ap option

Supplies parameters to the authenticate_parameters script and to authentication parameters.

Syntax

dbmlsync -ap "*parameters*,...**"** ...

Remarks

Use when you use the authenticate_parameters connection script or authentication parameters. 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*]

-ba 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 option” on page 139
- “-be option” on page 140
- “-bg option” on page 141

-bc 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, in the same dbmlsync command line as you create the 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 option" on page 138
- "-be option" on page 140
- "-bg option" on page 141

-be 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 304
- "MobiLink file-based download" [*MobiLink - Server Administration*]
- "-bc option" on page 139
- "-ba option" on page 138

-bg option

When creating a download file, this option creates a file that can be used with 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 option" on page 138
- "-bc option" on page 139
- "MobiLink generation numbers" [*MobiLink - Server Administration*]
- "Synchronizing new remotes" [*MobiLink - Server Administration*]

-c 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 fields of the connection window.

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

-d option

Drops conflicting locks to the remote database.

Syntax

dbmlsync -d ...

Remarks

During synchronization, unless the LockTables extended option is set to OFF, all tables involved in the publications being synchronized are locked to prevent any other processes from making changes. 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 115

-dc option

Restart a previously failed download.

Syntax

dbmlsync -dc ...

Remarks

By default, if MobiLink 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 start dbmlsync, it can more quickly complete the download. When you specify -dc, 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 use -dc, 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

- "Resuming failed downloads" [*MobiLink - Server Administration*]
- "ContinueDownload (cd) extended option" on page 191
- "sp_hook_dbmlsync_end" on page 273
- "DownloadReadSize (drs) extended option" on page 195

-dl option

Displays messages in the dbmlsync messages window or command prompt, and the message log file.

Syntax

dbmlsync -dl ...

Remarks

Normally when output is logged to a file, more messages are written to the log file than to the dbmlsync window. This option forces dbmlsync to write information normally only written to the file to the window as well. Using this option may have an effect on the speed of synchronization.

-do option

Disables scanning of offline transaction logs.

Syntax

dbmlsync -do ...

Remarks

If transaction log files for multiple databases are stored in a single directory, dbmlsync 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 dbmlsync with the -do option is used, dbmlsync does not attempt to scan any offline transaction logs. Therefore, dbmlsync with -do should be able to sync from a database that is stored with all the other databases in a single directory, if this database does not have any offline transaction log files.

If this option is used and if offline transaction logs are required, dbmlsync is not be able to sync.

Cannot be used with -x option.

-drs option

For restartable downloads, specifies the maximum amount of data 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 195
- "Resuming failed downloads" [*MobiLink - Server Administration*]
- "ContinueDownload (cd) extended option" on page 191
- "sp_hook_dbmlsync_end" on page 273
- "-dc option" on page 144

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -drs 100
```

-ds option

Performs a download-only synchronization.

Syntax

dbmlsync -ds ...

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 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 194
- "Download-only publications" on page 103

-e option

Specifies extended options.

Syntax

dbmlsync -e *extended-option***=***value*; ...

extended-option:
adr cd cr ctp dbs dir drs ds eh el ft hrt inc isc lt mem mn mp p pp sa sc sch scn st sv toc tor uo v vn vm vo vr vs vu

Parameters

Extended options can be specified by their long form or short form.

See "MobiLink SQL Anywhere client extended options" on page 183.

Remarks

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 sv=test applies to the synchronization of both pub1 and pub2.

```
dbmlsync -e "sv=test" -n pub1 -n pub2
```

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 option" on page 153
- "SYSSYNC system view" [*SQL Anywhere Server - SQL Reference*]
- "sp_hook_dbmlsync_set_extended_options" on page 292

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 option

Ignores errors that occur in hook functions.

Syntax

dbmlsync -eh ...

-ek option

Allows you to specify the encryption key for strongly encrypted databases directly on the command line.

Syntax

dbmlsync -ek *key* ...

Remarks

If you have a strongly encrypted 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 option

Prompt for the encryption key.

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 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 option

Specifies extended upload options.

Syntax

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 they follow. For example, on the following command line, the extended option sv=test applies only to the synchronization of pub2.

```
dbmlsync -n pub1 -n pub2 -eu "sv=test"
```

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 183.

For a complete list of extended options, see "MobiLink SQL Anywhere client extended options" on page 183.

-is option

Ignores scheduling instructions so that synchronization is immediate.

Syntax

dbmlsync -is ...

Remarks

Ignore extended options that schedule synchronization.

For information about scheduling, see "Scheduling synchronization" on page 121.

-k option (deprecated)

Shuts down dbmlsync when synchronization is finished. This option is deprecated. Use -qc instead.

Syntax

dbmlsync -k ...

See also

- "-qc option" on page 169

-l option

Lists available extended options.

Syntax

dbmlsync -l ...

Remarks

When used with the dbmlsync command line it shows you available extended options.

See also

- “MobiLink SQL Anywhere client extended options” on page 183

-mn 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 9.

See also

- “MobiLinkPwd (mp) extended option” on page 207
- “NewMobiLinkPwd (mn) extended option” on page 208
- “-mp option” on page 158

-mp 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 9.

See also

- "MobiLinkPwd (mp) extended option" on page 207
- "NewMobiLinkPwd (mn) extended option" on page 208
- "-mn option" on page 157

-n option

Specifies the publication(s) to synchronize.

Syntax

dbmlsync -n *pubname* ...

Remarks

Name of synchronization publication.

You can supply more than one -n option to synchronize more than one synchronization publication, however publication can only be specified once in a synchronization profile.

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, only the options set on the first publication in the list are used. Extended options set on subsequent publications 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`, it is possible that dbmlsync may 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 253.

-o option

Sends output to 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 option" on page 161
- "-ot option" on page 162

-os 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 logging output messages, 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*.dbr, 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 option" on page 160
- "-ot option" on page 162

-ot option

Deletes the contents of the message log 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 option" on page 160
- "-os option" on page 161

-p option

Disables logscan polling.

Syntax

dbmlsync -p ...

Remarks

To build an upload, dbmlsync must scan the transaction log. Usually it does this just before 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.

Logscan polling is on by default but only has an effect when synchronizations are scheduled using scheduling options or when sp_hook_dbmlsync_delay hook is used. When in effect, polling occurs at set intervals; by default this is 1 minute, but it can be changed with the dbmlsync -pp option.

This option is identical to the extended option DisablePolling=on.

See also

- "DisablePolling (p) extended option" on page 192
- "PollingPeriod (pp) extended option" on page 211
- "-pp option" on page 167

-pc option

Maintain a persistent connection to the MobiLink server between synchronizations.

Syntax

dbmlsync -pc+ ...

Remarks

When this option is specified, dbmlsync connects to the MobiLink server as usual, but it then keeps that connection open for use with 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 67.
- 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 option

Preload specified DLLs for Windows Mobile.

Syntax

dbmlsync -pd *dllname*;...

Remarks

When running dbmlsync 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, dbmlsync 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	mlcecc10.dll
RSA	mlcrsa10.dll
FIPS	mlcrsafips10.dll

You should specify multiple DLLs as a semicolon-separated list. For example:

```
-pd mlcrsafips10.dll;mlcrsa10.dll
```

-pi option

Pings a MobiLink server.

Syntax

dbmlsync -pi -c *connection_string* ...

Remarks

When you use -pi, 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-pi option with all the synchronization options you want to use to synchronize with dbmlsync. When -pi 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.

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

-pp 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 just before 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 211
- "DisablePolling (p) extended option" on page 192
- "-p option" on page 163

-q option

Starts the MobiLink synchronization client in a minimized window.

Syntax

dbmlsync -q ...

-qc 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 option" on page 160
- "-ot option" on page 162

-r 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 98.

-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.

-sc option

Specifies that dbmlsync should reload schema information before each synchronization.

Syntax

dbmlsync -sc...

Remarks

Prior to version 9.0, dbmlsync 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 in most cases the information does not change between synchronizations.

Starting with version 9.0, by default dbmlsync loads schema information only at startup. Specify -sc if you want the information to be loaded before every synchronization.

-sp 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 overrides 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*]

-tu 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. In most cases, only the transactions that were not successfully completed are sent at that time. In some cases, 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 option” [*MobiLink - Server Administration*]
- “Uploading data from self-referencing tables” [*MobiLink - Server Administration*]

-u option

Specifies the MobiLink user name.

Syntax

dbmlsync -u *ml_username* ...

Remarks

You can specify one user in 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.

-ui option

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

Syntax

mlsrv11 -c "*connection-string***" -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 window server isn't running, then dbmlsync starts in shell mode.

-uo 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 194
- "UploadOnly (uo) extended option" on page 221

-urc 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

- "Memory (mem) extended option" on page 205
- "For large uploads, estimate the number of rows" [*MobiLink - Server Administration*]

-ux 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 option" on page 168

-v 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.

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 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 222
- "VerboseHooks (vs) extended option" on page 223
- "VerboseMin (vm) extended option" on page 224
- "VerboseOptions (vo) extended option" on page 225
- "VerboseRowCounts (vn) extended option" on page 226
- "VerboseRowValues (vr) extended option" on page 227
- "-o option" on page 160
- "-ot option" on page 162

-wc option

Specifies a window class name.

Syntax

dbmlsync -wc *class-name* ...

Remarks

This option specifies a window class name that can be used to poke dbmlsync and wake it up 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 ActiveSync synchronization. The class name must be given when registering the application for use with ActiveSync synchronization.

This option applies only to Windows.

See also

- "Registering SQL Anywhere clients for ActiveSync" on page 119
- "Using ActiveSync synchronization" on page 117
- INFINITE keyword in "Schedule (sch) extended option" on page 212
- "Scheduling synchronization" on page 121

Example

```
dbmlsync -wc dbmlsync_$message_end...
```

-x option

Renames and restarts the transaction log after it has been scanned for outgoing messages.

Syntax

dbmlsync -x [*size* [**K** | **M** | **G**] ...

Remarks

The optional *size* means that the transaction log is renamed only if it is larger than the specified size. Use the suffix k, m or g to specify units of kilobytes, megabytes or gigabytes, respectively. The default size is 0.

In some circumstances, synchronizing data to a consolidated database can take the place of backing up remote databases, or renaming the transaction log when the database server is shut down.

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 [MobiLink client] [SQL Remote] [Replication Agent]" [*SQL Anywhere Server - Database Administration*]
- "CREATE EVENT statement" [*SQL Anywhere Server - SQL Reference*]

MobiLink SQL Anywhere client extended options

Contents

Introduction to dbmlsync extended options

Extended options can be specified on the dbmlsync 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_dbmlsync_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

Dbmlsync combines options stored in the database with those specified on the command line. If conflicting options are specified, dbmlsync 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_dbmlsync_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 113.

See also

- "-e option" on page 149
- "-eu option" on page 153
- "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 292

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
```

CommunicationAddress (adr) extended option

Specifies network protocol options for connecting to the MobiLink server.

Syntax

adr=*protocol-option*; ...

Parameters

See "MobiLink client network protocol option summary" on page 35.

Remarks

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.

If you are using the Redirector, see "Configuring MobiLink clients and servers for the Redirector" [*MobiLink - Server Administration*].

This option has a short form and long form: you can use **adr** or **CommunicationAddress**.

This option can also be stored in the database using the SQL statement that creates or alters a publication, subscription, or user. For more information, see:

- "CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*]
- "CREATE SYNCHRONIZATION USER statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*]

Use the CommunicationType extended option to specify the type of network protocol.

See "CommunicationType (ctp) extended option" on page 189.

See also

- "MobiLink client network protocol options" on page 33
- "Configuring MobiLink clients and servers for the Redirector" [*MobiLink - Server Administration*]

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*; ...

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.

This option has a short form and long form: you can use **ctp** or **CommunicationType**.

See also

- "Encrypting MobiLink client/server communications" [*SQL Anywhere Server - Database Administration*]
- "CommunicationAddress (adr) extended option" on page 187

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "ctp=https"
```

To store the CommunicationType in the database, you can use an extended option or you can use the TYPE clause. For example,

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   TYPE 'tcpip'
```

ConflictRetries (cr) extended option

Specifies the number of retries if the download fails because of conflicts.

Syntax

cr=*number*; ...

Remarks

When the extended option LockTables is set to OFF (preventing dbmlsync from obtaining locks on the tables being synchronized), 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. This option controls the number of retries that are performed.

This option is useful only if the LockTables option is OFF, which is not the default.

The default is **-1** (retries should continue indefinitely).

This option has a short form and long form: you can use **cr** or **ConflictRetries**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

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** }; ...

Remarks

If MobiLink 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 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 restartable download fails.

You can also specify restartable downloads for SQL Anywhere remote databases with the -dc option or with the sp_hook_dbmlsync_end hook.

This option has a short form and long form: you can use **cd** or **ContinueDownload**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

See also

- "Resuming failed downloads" [*MobiLink - Server Administration*]
- "sp_hook_dbmlsync_set_extended_options" on page 292
- "-dc option" on page 144
- "sp_hook_dbmlsync_end" on page 273

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** }; ...

Remarks

To build an upload, dbmlsync must scan the transaction log. Usually it does this just before synchronization. However, when synchronizations are scheduled, dbmlsync by default scans the log in the time between scheduled synchronizations; and when the sp_hook_dbmlsync_delay hook is used, dbmlsync by default scans the log in the pause that occurs just before synchronization. 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**.

This option has a short form and long form: you can use **p** or **DisablePolling**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

See also

- "PollingPeriod (pp) extended option" on page 211
- "-p option" on page 163
- "-pp option" on page 167

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';
```

DownloadBufferSize (dbs) extended option

Specifies the size of the download buffer.

Syntax

dbs=*number*[**K** | **M**]; ...

Remarks

The buffer size is specified in units of bytes. Use the suffix k or m to specify units of kilobytes or megabytes, respectively.

If you set this option to 0, dbmlsync does not buffer the download. If this option is greater than 0, the entire download stream is read from the communication stream with the MobiLink server before it is applied to the remote database. If the download stream fits in the space specified by the option then it is held entirely in memory; otherwise some of it is written to a temporary file.

If the setting is greater than 0 but less than 4 KB, dbmlsync uses a 4 KB buffer size and issues a warning. The default is **32k** on Windows Mobile, and **1m** on all other operating systems.

This option has a short form and long form: you can use **dbs** or **DownloadBufferSize**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "dbs=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 dbs='32k';
```

DownloadOnly (ds) extended option

Specifies that synchronization should be download-only.

Syntax

ds={ **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 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** (full synchronization of both upload and download).

This option has a short form and long form: you can use **ds** or **DownloadOnly**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

See also

- "-ds option" on page 148
- "Download-only publications" on page 103
- "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**]; ...

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.

Dbmlsync reads the download in chunks. The DownloadReadSize 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 range 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.

This option has a short form and long form: you can use **drs** or **DownloadReadSize**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

See also

- "-drs option" on page 147
- "Resuming failed downloads" [*MobiLink - Server Administration*]
- "ContinueDownload (cd) extended option" on page 191
- "sp_hook_dbmlsync_end" on page 273
- "-dc option" on page 144

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -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**]; ...

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.

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 logs the last part of the newly generated error and log messages 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 option” on page 180 and the -e verbose options:

- “Verbose (v) extended option” on page 222
- “VerboseHooks (vs) extended option” on page 223
- “VerboseMin (vm) extended option” on page 224
- “VerboseOptions (vo) extended option” on page 225
- “VerboseRowCounts (vn) extended option” on page 226
- “VerboseRowValues (vr) extended option” on page 227
- “VerboseUpload (vu) extended option” on page 228

The default is **32K**.

This option has a short form and long form: you can use **el** or **ErrorLogSendLimit**.

You can also store extended options in the database. For more information about dbmlsync extended options, see “Introduction to dbmlsync extended options” on page 185.

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** }; ...

Remarks

The default is **ON**.

This option has a short form and long form: you can use **ft** or **FireTriggers**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

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**]; ...

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 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_dbmlsync_log_rescan stored procedure.

This option has a short form and long form: you can use **hrt** or **HoverRescanThreshold**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

See also

- "sp_hook_dbmlsync_log_rescan" on page 276

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -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** }; ...

Remarks

The default is **OFF**.

This option has a short form and long form: you can use **eh** or **IgnoreHookErrors**.

This option is equivalent to the dbmlsync -eh option.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -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 scheduling settings should be ignored.

Syntax

isc={ **ON** | **OFF** }; ...

Remarks

If set to ON, dbmlsync ignores any scheduling information that is specified in extended options and synchronizes immediately. The default is **OFF**.

This option is equivalent to the dbmlsync -is option.

This option has a short form and long form: you can use **isc** or **IgnoreScheduling**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

See also

- "Scheduling synchronization" on page 121
- "Schedule (sch) extended option" on page 212

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**]; ...

Remarks

This option specifies a minimum incremental scan volume in units of bytes. Use the suffix k or m to specify units of kilobytes or megabytes, respectively.

When this option is specified, uploads are sent to MobiLink in one or more parts. This could be useful if a site has difficulty maintaining a connection for long enough to complete the full upload. When the option is not set, uploads are sent as a single unit.

The value of this option specifies, very approximately, the size of each upload part. 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.

This option has a short form and long form: you can use **inc** or **Increment**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

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** }; ...

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 ON.
- If there are passthrough scripts that were downloaded in the previous synchronization and these scripts need to be executed automatically in the current synchronization, the synchronization tables are locked with SHARE or EXCLUSIVE depending on the passthrough scripts' requirement.

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 115.

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 235.

This option has a short form and long form: you can use **lt** or **LockTables**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

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';
```

Memory (mem) extended option

Specifies a cache size that is used by dbmlsync when building the upload.

Syntax

mem=*number*[**K** | **M**]; ...

Remarks

Specifies the size of the cache used for building the upload, in units of bytes. A larger cache means that dbmlsync can keep more pages of data in memory, reduce the number of disk reads/writes, and improve performance.

Use the suffix k or m to specify units of kilobytes or megabytes, respectively. The default is **1M**.

This option has a short form and long form: you can use **mem** or **Memory**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

See also

- "-urc option" on page 178
- "Performance tips" [*MobiLink - Server Administration*]

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "mem=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 mem='2m';
```

MirrorLogDirectory (mld) extended option

Specifies the location of old transaction log mirror files so that they can be deleted.

Syntax

mld=*filename*; ...

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.

This option has a short form and long form: you can use **mld** or **MirrorLogDirectory**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

See also

- "delete_old_logs option [MobiLink client] [SQL Remote] [Replication Agent]" [*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*; ...

Remarks

Specifies the password used to connect. This password should be the correct password for the MobiLink user whose subscriptions are being synchronized. This user may be specified with the dbmlsync -u option. The default is null.

If the MobiLink user already has a password, use the extended option **-e mn** to change it.

This option has a short form and long form: you can use **mp** or **MobiLinkPwd**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

See also

- "NewMobiLinkPwd (mn) extended option" on page 208
- "-mn option" on page 157
- "-mp option" on page 158

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='SQL';
```

NewMobiLinkPwd (mn) extended option

Specifies a new password.

Syntax

mn=*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.

This option has a short form and long form: you can use **mn** or **NewMobiLinkPwd**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

See also

- "MobiLinkPwd (mp) extended option" on page 207
- "-mn option" on page 157
- "-mp option" on page 158

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 mn='SQL';
```

NoSyncOnStartup (nss) extended option

Prevents dbmlsync from synchronizing on startup when a scheduling option would otherwise cause that to happen.

Syntax

nss={ **on** | **off** }; ...

Remarks

This option has an effect only when a schedule 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.

This option has a short form and long form: you can use **nss** or **NoSyncOnStartup**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

See also

- "Schedule (sch) extended option" on page 212
- "Scheduling synchronization" on page 121

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*; ...

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.

This option has a short form and long form: you can use **dir** or **OfflineDirectory**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

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**]; ...

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_dbmlsync_delay hook.

For an explanation of logscan polling, see "DisablePolling (p) extended option" on page 192.

This option is identical to **dbmlsync -pp**.

This option has a short form and long form: you can use **pp** or **PollingPeriod**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

See also

- "DisablePolling (p) extended option" on page 192
- "-pp option" on page 167
- "-p option" on page 163
- "sp_hook_dbmlsync_delay" on page 253

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -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 : { **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**

Parameters

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 209.

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 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. While it waits, dbmlsync runs and scans the log periodically. You can use the dbmlsync extended option NoSyncOnStartup to avoid the initial synchronization.

For more information, see "NoSyncOnStartup (nss) extended option" on page 209.

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 option" on page 181.

Remarks

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.

This option has a short form and long form: you can use **sch** or **Schedule**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

The schedule option syntax is the same when used in the synchronization SQL statements and in the dbmlsync command line.

The IgnoreScheduling extended option and the -is option instruct dbmlsync to ignore scheduling, so that synchronization is immediate. For more information, see "IgnoreScheduling (isc) extended option" on page 201.

For more information about scheduling, see "Scheduling synchronization" on page 121.

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -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.

Syntax

sv=*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**.

This option has a short form and long form: you can use **sv** or **ScriptVersion**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "sv=SyaAd001"
```

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.

Syntax

scn={ **ON** | **OFF** }; ...

Remarks

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, you should set this option. This is the only use of the column names that are sent by this option.

See "Direct row handling" [*MobiLink - Server Administration*].

The default is **OFF**.

This option has a short form and long form: you can use **scn** or **SendColumnNames**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

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** }; ...

Remarks

It is recommended that SendDownloadAck be set to OFF. If the download fails, the remote uploads the same timestamp over again, and no data is lost. This option is used to change the behavior of server-side scripts. See "nonblocking_download_ack connection event" [*MobiLink - Server Administration*].

For more information about improving performance by turning off the download acknowledgement, see "Use non-blocking download acknowledgement" [*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**.

This option has a short form and long form: you can use **sa** or **SendDownloadACK**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -e "sa=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 sa='off';
```

SendTriggers (st) extended option

Specifies that trigger actions should be sent on upload.

Syntax

st={ **ON** | **OFF** }; ...

Remarks

Cascaded deletes are also considered trigger actions.

The default is **OFF**.

This option has a short form and long form: you can use **st** or **SendTriggers**.

If two publications are overlapping, that is they both contain one or more of the same tables, then both publications must be synchronized with the same setting for the SendTriggers option.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

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*; ...

tables = *table-name* [,*table-name*], ...

Remarks

This option allows you to specify the order of tables on the remote database that are to be uploaded. Specify *tables* as a comma-separated list. 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.

There are no cases where this option must be used. It is provided for users who want to ensure that tables are uploaded in a specific order.

This option has a short form and long form: you can use **tor** or **TableOrder**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

See also

- "TableOrderChecking (toc) extended option" on page 220
- "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

When you specify TableOrder, determines whether dbmlsync should check that no table is uploaded before another table on which it has a foreign key.

Syntax

tor={ 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 does not follow the rule that no table is uploaded before one on which it has a foreign key.

This option is only useful when the TableOrder extended option is specified.

The default is **ON**.

This option has a short form and long form: you can use **toc** or **TableOrderChecking**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

See also

- "TableOrder (tor) extended option" on page 218
- "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** }; ...

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, 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*].

The default is **OFF**.

This option has a short form and long form: you can use **uo** or **UploadOnly**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

See also

- "Upload-only and download-only synchronizations" [*MobiLink - Server Administration*]
- "DownloadOnly (ds) extended option" on page 194

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** }; ...

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 option" on page 180.

The default is **OFF**.

This option has a short form and long form: you can use **v** or **Verbose**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

See also

- "VerboseHooks (vs) extended option" on page 223
- "VerboseMin (vm) extended option" on page 224
- "VerboseOptions (vo) extended option" on page 225
- "VerboseRowCounts (vn) extended option" on page 226
- "VerboseRowValues (vr) extended option" on page 227
- "VerboseUpload (vu) extended option" on page 228

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** }; ...

Remarks

This option is identical to **dbmlsync -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 option” on page 180.

The default is **OFF**.

This option has a short form and long form: you can use **vs** or **VerboseHooks**.

You can also store extended options in the database. For more information about dbmlsync extended options, see “Introduction to dbmlsync extended options” on page 185.

See also

- “Event hooks for SQL Anywhere clients” on page 235
- “Verbose (v) extended option” on page 222
- “VerboseMin (vm) extended option” on page 224
- “VerboseOptions (vo) extended option” on page 225
- “VerboseRowCounts (vn) extended option” on page 226
- “VerboseRowValues (vr) extended option” on page 227
- “VerboseUpload (vu) extended option” on page 228

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -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 }; ...

Remarks

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 option" on page 180.

The default is **OFF**.

This option has a short form and long form: you can use **vm** or **VerboseMin**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

See also

- "Verbose (v) extended option" on page 222
- "Verbose (v) extended option" on page 222
- "VerboseOptions (vo) extended option" on page 225
- "VerboseRowCounts (vn) extended option" on page 226
- "VerboseRowValues (vr) extended option" on page 227
- "VerboseUpload (vu) extended option" on page 228

Example

The following dbmlsync command line illustrates how you can set this option when you start dbmlsync:

```
dbmlsync -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** }; ...

Remarks

This option is identical to **dbmlsync -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 option" on page 180.

The default is **OFF**.

This option has a short form and long form: you can use **vo** or **VerboseOptions**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

See also

- "Verbose (v) extended option" on page 222
- "Verbose (v) extended option" on page 222
- "VerboseMin (vm) extended option" on page 224
- "VerboseRowCounts (vn) extended option" on page 226
- "VerboseRowValues (vr) extended option" on page 227
- "VerboseUpload (vu) extended option" on page 228

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** }; ...

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 option" on page 180.

The default is **OFF**.

This option has a short form and long form: you can use **vn** or **VerboseRowCounts**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

See also

- "Verbose (v) extended option" on page 222
- "Verbose (v) extended option" on page 222
- "VerboseMin (vm) extended option" on page 224
- "VerboseOptions (vo) extended option" on page 225
- "VerboseRowValues (vr) extended option" on page 227
- "VerboseUpload (vu) extended option" on page 228

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** }; ...

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 option" on page 180.

The default is **OFF**.

This option has a short form and long form: you can use **vr** or **VerboseRowValues**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

See also

- "Verbose (v) extended option" on page 222
- "Verbose (v) extended option" on page 222
- "VerboseMin (vm) extended option" on page 224
- "VerboseOptions (vo) extended option" on page 225
- "VerboseRowCounts (vn) extended option" on page 226
- "VerboseUpload (vu) extended option" on page 228

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 steam should be logged.

Syntax

vu={ **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 option" on page 180.

The default is **OFF**.

This option has a short form and long form: you can use **vu** or **VerboseUpload**.

You can also store extended options in the database. For more information about dbmlsync extended options, see "Introduction to dbmlsync extended options" on page 185.

See also

- "Verbose (v) extended option" on page 222
- "Verbose (v) extended option" on page 222
- "VerboseMin (vm) extended option" on page 224
- "VerboseOptions (vo) extended option" on page 225
- "VerboseRowCounts (vn) extended option" on page 226
- "VerboseRowValues (vr) extended option" on page 227

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

Contents

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*]
- "START SYNCHRONIZATION DELETE statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*]
- "STOP SYNCHRONIZATION DELETE statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*]

UltraLite clients

See "UltraLite SQL statements" [*UltraLite - Database Management and Reference*].

MobiLink synchronization profiles

Contents

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.

Once you have created a synchronization profile and want to use it, use the dbmlsync -sp option and specify the synchronization profile name. The options specified in the synchronization profile are merged with the command line options you specified. If equivalent options are specified on the command line and in the synchronization profile, then the options on the command line overrides those specified in the profile. See "-sp option" on page 172.

Synchronization profiles can be created, altered and dropped using the following statements:

- "ALTER SYNCHRONIZATION PROFILE statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*]
- "CREATE SYNCHRONIZATION PROFILE statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*]
- "DROP SYNCHRONIZATION PROFILE statement [MobiLink]" [*SQL Anywhere Server - SQL Reference*]

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**	**Valid values**	**Description**
AuthParms	ap	String	Supplies parameters to the authenticate_parameters script and to authentication parameters. See "-ap option" on page 137.
ApplyDnldFile	ba	String	Applies a download file. See "-ba option" on page 138.
ContinueDownload	dc	Boolean	Restarts a previously failed download. See "-dc option" on page 144.
CreateDnldFile	bc	String	Creates a download file. See "-bc option" on page 139.
DnldFileExtra	be	String	When creating a download file, this option specifies an extra string to be included in the file. See "-be option" on page 140.
DownloadOnly	ds	Boolean	Performs a download-only synchronization. See "-ds option" on page 148.
DownloadReadSize	drs	Integer	For restartable downloads, specifies the maximum amount of data that may need to be resent after a communications failure. See "-drs option" on page 147.

Long option name	Short name	Valid values	Description
ExtOpt	e	String	Specifies extended options. See "-e option" on page 149.
IgnoreHookErrors	eh	Boolean	Ignores errors that occur in hook functions. See "-eh option" on page 150.
IgnoreScheduling	is	Boolean	Ignores scheduling instructions so that synchronization is immediate. See "-is option" on page 154.
KillConnections	d	Boolean	Drops conflicting locks to the remote database. See "-d option" on page 143.
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 "-x option" on page 182.
MobiLinkPwd	mp	String	Supplies the password of the MobiLink user. See "-mp option" on page 158.
MLUser	u	String	Specifies the MobiLink user name. See "-u option" on page 175.
NewMLPassword	mn	String	Supplies a new password for the MobiLink user. Use this option when you want to change an existing password. See "-mn option" on page 157.
Ping	pi	Boolean	Pings a MobiLink server to confirm communications between the client and MobiLink. See "-pi option" on page 166.
Publication	n	String	Specifies the publications(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 "-n option" on page 159.
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 "-r option" on page 170.
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 "-r option" on page 170.

Long option name	Short name	Valid values	Description
Transactiona-lUpload	tu	Boolean	Specifies that each transaction on the remote database should be uploaded as a separate transaction within one synchronization. See "-tu option" on page 173.
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 "-bg option" on page 141.
UploadOnly	uo	Boolean	Specifies that synchronization only includes an upload, and that no downloads occur. See "-uo option" on page 177.
UploadRowCnt	urc	Integer	Specifies an estimate of the number of rows to be uploaded in a synchronization. See "-urc option" on page 178.
Verbosity		String (a comma separated list of options)	Controls dbmlsync verbosity. Similar to the "-v option" on page 180. 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: ● 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 ● HOOK - equivalent to -vs

Event hooks for SQL Anywhere clients

Contents

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 prior to the synchronization and another to copy the data back afterwards.

> **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.
>
> Also, changing connection settings in a hook may produce unexpected results. If you need to change connection settings in a hook, the hook should restore the old value before the hook completes.

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 123.

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.

Each hook is provided with parameter values that you can use when you implement the procedure. In some cases, 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 prior to 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_dbmlsync_logscan_begin are synchronized in the current synchronization session; after that point, changes are synchronized in the next session.

```
sp_hook_dbmlsync_abort
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)
for each download segment
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
```

For more information about upload options, see "-tu option" on page 173 and "Increment (inc) extended option" on page 202.

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.
- Don't change connection settings. Changing connection settings in a hook may produce unexpected results. If you need to change connection settings in a hook, the hook should restore the old value before the hook completes.
- The event-hook connection calls the stored procedures without qualifying them by owner. The stored procedures must therefore be owned by either the user name employed on the dbmlsync connection (typically a user with REMOTE DBA authority), or a group ID of which the dbmlsync user is a member.
- A remote database can only have one instance of each hook. Do not create multiple instances of a hook that have different owners.
- Hook procedures must be created by a user with DBA authority.
- 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 prior to the point in time where the *_begin hook would normally be called.

#hook_dict table

Immediately before a hook is called, dbmlsync 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 dbmlsync 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 dbmlsync.

Each hook receives parameter values. In some cases, 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, for the following dbmlsync command line, when the sp_hook_dbmlsync_abort hook is called, the #hook_dict table contains the following rows:

```
dbmlsync -c 'dsn=MyDsn' -n pub1,pub2 -u MyUser
```

#hook_dict row	Value
publication_0	pub1
publication_1	pub2
MobiLink user	MyUser

#hook_dict row	Value
Abort synchronization	false

Your abort 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, your hook 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 the #hook_dict table to pass arguments. 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 syncdef VARCHAR(150);
   SELECT '>>>syncdef = ' || value INTO syncdef
      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_com_error
- sp_hook_dbmlsync_download_fatal_sql_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 263
- “sp_hook_dbmlsync_download_ri_violation” on page 265

Handling MobiLink connection failures

The sp_hook_dbmlsync_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, dbmlsync calls the sp_hook_dbmlsync_communication_error and sp_hook_dbmlsync_all_error hooks.

See “sp_hook_dbmlsync_ml_connect_failed” on page 285.

Handling dbmlsync errors

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, sp_hook_dbmlsync_all_error is called. This hook is useful for logging all errors that occur.

See:

- "sp_hook_dbmlsync_communication_error" on page 250
- "sp_hook_dbmlsync_sql_error" on page 297
- "sp_hook_dbmlsync_misc_error" on page 282
- "sp_hook_dbmlsync_all_error" on page 245

If you want to restart a synchronization in response to an error, you can use the user state parameter in sp_hook_dbmlsync_end.

See "sp_hook_dbmlsync_end" on page 273.

Ignoring errors

By default, synchronization stops when an error is encountered in an event hook procedure. You can instruct the dbmlsync utility to ignore errors that occur in event hook procedures by supplying the -eh option.

See "IgnoreHookErrors (eh) extended option" on page 200.

sp_hook_dbmlsync_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 #hook_dict table to **true**, then dbmlsync terminates immediately after the event.
publication_*n* (in)	publication	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* 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.

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.

If the hook requests an abort by setting the abort synchronization value to true, the exit code is passed to the sp_hook_dbmlsync_process_exit_code hook. If no sp_hook_dbmlsync_process_exit_code hook is defined, the exit code is used as the exit code for the program.

Actions of this procedure are committed immediately after execution.

See also

- "Synchronization event hook sequence" on page 237
- "sp_hook_dbmlsync_process_exit_code" on page 288

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 dbmlsync to have an exit code of 0. The other reason indicates an error condition, so you want dbmlsync to have a non-zero exit code. You could achieve this with an sp_hook_dbmlsync_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_dbmlsync_all_error

Use this stored procedure to process dbmlsync error messages of all types. For example, you can implement the sp_hook_dbmlsync_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	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. 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.
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 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 the hook changes the value of the row, the new value is used in the next hook call. When you use this hook to pass state information to the sp_hook_dbmlsync_end hook, you can cause the sp_hook_dbmlsync_end hook to perform actions such as retrying the synchronization.

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* rows are set in the #hook_dict table.

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.

By default on Windows Mobile devices, synchronization tables are locked in exclusive mode, which means that this hook cannot successfully execute if it requires access to any of the synchronization tables. It also cannot execute if it needs to access synchronization tables and you set the dbmlsync extended option LockTables to EXCLUSIVE. See "LockTables (lt) extended option" on page 203.

Actions of this procedure are committed immediately after the hook completes.

See also

- "Handling errors and warnings in event hook procedures" on page 241
- "sp_hook_dbmlsync_communication_error" on page 250
- "sp_hook_dbmlsync_misc_error" on page 282
- "sp_hook_dbmlsync_sql_error" on page 297

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 273.

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_*n* (in)	publication	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. 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.

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 237

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_name"        varchar(128) NOT NULL ,
   "ml_user"             varchar(128) NULL ,
   "event_time"          timestamp NULL,
   "table_name"          varchar(128) NULL ,
   "upsert_count"        varchar(128) NULL ,
   "delete_count"        varchar(128) NULL ,
   "exit_code"           integer NULL ,
   "status_retval"       varchar(128) NULL ,
   "pubs"                 varchar(128) NULL ,
   "sync_descr "          varchar(128) NULL ,
    PRIMARY KEY ("event_id"),
)
```

The following example compiles a list of publications. It logs the list of publications and other synchronization information at the beginning of the synchronization process.

```
CREATE PROCEDURE sp_hook_dbmlsync_begin ()
BEGIN

    DECLARE pubs_list VARCHAR(1024);
    DECLARE temp_str VARCHAR(128);
    DECLARE qry VARCHAR(128);

-- insert publication list into pubs_list
    SELECT LIST(value) INTO pubs_list
     FROM #hook_dict
     WHERE name LIKE 'publication_%';

-- log publication and synchronization information
    INSERT INTO SyncLog(event_name,ml_user,pubs,event_time)
     SELECT 'dbmlsync_begin',#hook_dict.value,pubs_list,CURRENT TIMESTAMP
     FROM #hook_dict
     WHERE name='MobiLink user';
END
```

sp_hook_dbmlsync_communication_error

Use this stored procedure to process communications errors.

Rows in #hook_dict table

Name	Value	Description
publication_*n* (in)	publication	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. 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.
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 the hook changes the value of the row, the new value is used in the next hook call. When you use this hook to pass state information to the sp_hook_dbmlsync_end hook, you can cause the _end hook to perform actions such as retrying the synchronization.
stream error code (in)	integer	The error reported by the stream.
system error code (in)	integer	A system-specific error code.

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* rows 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*].

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.

By default on Windows Mobile devices, synchronization tables are locked in exclusive mode, which means that this hook cannot successfully execute if it requires access to any of the synchronization tables. It also cannot execute if it needs to access synchronization tables and you set the dbmlsync extended option LockTables to EXCLUSIVE. See "LockTables (lt) extended option" on page 203.

Actions of this procedure are committed immediately after execution.

See also

- "Handling errors and warnings in event hook procedures" on page 241
- "sp_hook_dbmlsync_all_error" on page 245
- "sp_hook_dbmlsync_misc_error" on page 282
- "sp_hook_dbmlsync_sql_error" on page 297

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_dbmlsync_communication_error to log communication errors.

```
CREATE PROCEDURE sp_hook_dbmlsync_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_dbmlsync_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. 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_*n* (in)	publication	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. 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.

Remarks

If a procedure of this name exists, it is called before **sp_hook_dbmlsync_begin** at the beginning of the synchronization process.

Actions of this procedure are committed immediately after execution.

See also

- “Synchronization event hook sequence” on page 237
- “Initiating synchronization with event hooks” on page 122
- “sp_hook_dbmlsync_download_end” on page 259

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_dbmlsync_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_dbmlsync_end hook you can mark the high priority row as processed:

```
CREATE PROCEDURE sp_hook_dbmlsync_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;
```

See "sp_hook_dbmlsync_end" on page 273.

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_*n* (in)	publication	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. 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.

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 237

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_name"        VARCHAR(128) NOT NULL ,
   "ml_user"             VARCHAR(128) NULL ,
   "event_time"          TIMESTAMP NULL,
   "table_name"          VARCHAR(128) NULL ,
   "upsert_count"        VARCHAR(128) NULL ,
   "delete_count"        VARCHAR(128) NULL ,
   "exit_code"           INTEGER NULL ,
   "status_retval"       VARCHAR(128) NULL ,
   "pubs"                 VARCHAR(128) NULL ,
   "sync_descr "          VARCHAR(128) NULL ,
    PRIMARY KEY ("event_id"),
);
```

The following example compiles a list of publications. It logs the list of publications and other synchronization information at the beginning of the download stage of the synchronization.

```
CREATE PROCEDURE sp_hook_dbmlsync_download_begin ()
BEGIN
```

```
    DECLARE pubs_list VARCHAR(1024);
    DECLARE temp_str VARCHAR(128);
    DECLARE qry VARCHAR(128);

-- insert publication list into pubs_list
    SELECT LIST(value) INTO pubs_list
     FROM #hook_dict
     WHERE name LIKE 'publication_%';

-- log publication and synchronization information
    INSERT INTO SyncLog(event_name,ml_user,pubs,event_time)
     SELECT 'dbmlsync_download_begin',#hook_dict.value,
      pubs_list,CURRENT TIMESTAMP
     FROM #hook_dict
     WHERE name='MobiLink user';
END;
```

sp_hook_dbmlsync_download_com_error (deprecated)

Use this stored procedure to add custom actions when communications errors occur while reading the download sent by the MobiLink server.

This hook is deprecated. See "Handling errors and warnings in event hook procedures" on page 241.

Rows in #hook_dict table

Name	Value	Description
table name (in)	table name	The table to which operations were being applied when the error occurred. The value is an empty string if dbmlsync is unable to identify the table.
publication_*n* (in)	publication	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. 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.

Remarks

If a procedure of this name exists, it is invoked when a communication error is detected during the download phase of synchronization. The download is then terminated.

This procedure executes on a separate connection, so that failures can be logged. Otherwise, the action of logging would be rolled back along with the synchronization actions. If dbmlsync cannot establish a separate connection, the procedure is not called.

By default on Windows Mobile devices, synchronization tables are locked in exclusive mode, which means that this hook cannot successfully execute if it requires access to any of the synchronization tables. It also cannot execute if it needs to access synchronization tables and you set the dbmlsync extended option LockTables to EXCLUSIVE. See "LockTables (lt) extended option" on page 203.

Actions of this procedure are committed immediately after execution.

See also

- "Synchronization event hook sequence" on page 237

Examples

Assume you use the following table to log communication errors.

```
CREATE TABLE SyncLogComErrorTable
(
   " user_name "  VARCHAR(255) NOT NULL ,
   " event_time "   TIMESTAMP NOT NULL ,
);
```

The following example logs the MobiLink user and current time stamp when communications errors occur while reading the download sent by the MobiLink server. The information is stored on the SyncLogComErrorTable table on the remote database.

```
CREATE PROCEDURE sp_hook_dbmlsync_download_com_error ()
BEGIN
 INSERT INTO SyncLogComErrorTable (user_name, event_time)
  SELECT #hook_dict.value, CURRENT TIMESTAMP
  FROM #hook_dict
  WHERE name = 'MobiLink user';
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	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. 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.

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 237
- “Initiating synchronization with event hooks” on page 122
- “sp_hook_dbmlsync_delay” on page 253

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_name"       VARCHAR(128) NOT NULL ,
   "ml_user"            VARCHAR(128) NULL ,
   "event_time"         TIMESTAMP NULL,
   "table_name"         VARCHAR(128) NULL ,
   "upsert_count"       VARCHAR(128) NULL ,
   "delete_count"       VARCHAR(128) NULL ,
   "exit_code"          INTEGER NULL ,
   "status_retval"      VARCHAR(128) NULL ,
   "pubs"                VARCHAR(128) NULL ,
   "sync_descr "         VARCHAR(128) NULL ,
    PRIMARY KEY ("event_id"),
)
```

The following example compiles a list of publications. It logs the list of publications and other synchronization information at the end of the download stage of a synchronization.

```
CREATE PROCEDURE sp_hook_dbmlsync_download_end ()
BEGIN

    DECLARE pubs_list VARCHAR(1024);
    DECLARE temp_str VARCHAR(128);
    DECLARE qry VARCHAR(128);

-- insert publication list into pubs_list
    SELECT LIST(value) INTO pubs_list
     FROM #hook_dict
     WHERE name LIKE 'publication_%';

-- log publication and synchronization information
    INSERT INTO SyncLog(event_name,ml_user,pubs,event_time)
     SELECT 'dbmlsync_download_end',#hook_dict.value,
      pubs_list,CURRENT TIMESTAMP
     FROM #hook_dict
     WHERE name='MobiLink user';
END
```

sp_hook_dbmlsync_download_fatal_sql_error (deprecated)

Take action when a synchronization download is about to be rolled back because of a database error.

This hook is deprecated. See "Handling errors and warnings in event hook procedures" on page 241.

Rows in #hook_dict table

Name	Value	Description
table name (in)	table name	The table to which operations were being applied when the error occurred. The value is an empty string if dbmlsync is unable to identify the table.
SQL error code (in)	SQL error code	Identifies the SQL error code returned by the database when the operation failed.
publication_*n* (in)	publication	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. 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.

Remarks

If a procedure of this name exists, it is called immediately before a synchronization download is rolled back because of a database error. This occurs whenever a SQL error is encountered that cannot be ignored, or when the sp_hook_dbmlsync_download_SQL_error hook has already been called and has chosen not to ignore the error.

This procedure executes on a separate connection, so that failures can be logged. Otherwise, the action of logging would be rolled back along with the synchronization actions. If dbmlsync cannot establish a separate connection, the procedure is not called.

By default on Windows Mobile devices, synchronization tables are locked in exclusive mode, which means that this hook cannot successfully execute if it requires access to any of the synchronization tables. It also cannot execute if it needs to access synchronization tables and you set the dbmlsync extended option LockTables to EXCLUSIVE. See "LockTables (lt) extended option" on page 203.

Actions of this procedure are committed immediately after execution.

See also

- “Synchronization event hook sequence” on page 237
- “sp_hook_dbmlsync_download_sql_error (deprecated)” on page 267

Examples

Assume you use the following table to log SQL errors.

```
CREATE TABLE "DBA"."SyncLogComErrorTable"
(
   " error_code "       VARCHAR(255) NOT NULL ,
   " event_time "       TIMESTAMP NOT NULL ,
);
```

The following example logs the SQL error code and current time stamp when SQL errors occur while reading the download. The information is stored in SyncLogSQLErrorTable on the remote database.

```
CREATE PROCEDURE sp_hook_dbmlsync_download_fatal_sql_error ()
BEGIN
 INSERT INTO SyncLogSQLErrorTable (error_code, event_time)
  SELECT #hook_dict.value, CURRENT TIMESTAMP
  FROM #hook_dict
  WHERE name = 'SQL error code';
END;
```

sp_hook_dbmlsync_download_log_ri_violation

Logs referential integrity violations in the download process.

Rows in #hook_dict table

Name	Value	Description
publication_*n* (in)	publication	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. 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.
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.

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, dbmlsync checks for RI violations. If it finds any, it identifies a foreign key that has an RI violation and calls sp_hook_dbmlsync_download_log_ri_violation (if it is implemented). It then calls sp_hook_dbmlsync_download_ri_conflict (if it is implemented). If there is still a conflict, dbmlsync 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 dbmlsync 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.

By default on Windows Mobile devices, synchronization tables are locked in exclusive mode, which means that this hook cannot successfully execute if it requires access to any of the synchronization tables. It also cannot execute if it needs to access synchronization tables and you set the dbmlsync extended option LockTables to EXCLUSIVE. See "LockTables (lt) extended option" on page 203.

Do not attempt to use this hook to correct RI violation problems. It should be used for logging only. Use sp_hook_dbmlsync_download_ri_violation to resolve RI violations.

See also

- "sp_hook_dbmlsync_download_ri_violation" on page 265
- "Synchronization event hook sequence" on page 237

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 LogRIErrorTable on the remote database.

```
CREATE PROCEDURE sp_hook_dbmlsync_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_dbmlsync_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	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. 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.
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.

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 dbmlsync deletes the rows that are causing the conflict.

After the download is complete, but before it is committed, dbmlsync checks for RI violations. If it finds any, it identifies a foreign key that has an RI violation and calls sp_hook_dbmlsync_download_log_ri_violation (if it is implemented). It then calls sp_hook_dbmlsync_download_ri_conflict (if it is implemented). If there is still a conflict, dbmlsync 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 dbmlsync 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_dbmlsync_download_log_ri_violation" on page 263

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;
```

sp_hook_dbmlsync_download_sql_error (deprecated)

Handle database errors that occur while applying the download sent by the MobiLink server.

This hook is deprecated. See "Handling errors and warnings in event hook procedures" on page 241.

Rows in #hook_dict table

Name	Value	Description
table name (in)	table name	The table to which operations were being applied when the error occurred. The value is an empty string if dbmlsync is unable to identify the table.
continue (in|out)	**true** | **false**	Indicates whether the error should be ignored and synchronization should continue. This parameter should be set to **false** to call the sp_hook_dbmlsync_download_fatal_sql_error hook and stop synchronization. If you set this parameter to **true**, dbmlsync ignores the error and continues with synchronization, which may result in data loss.
SQL error code (in)	SQL error code	Identifies the SQL error code returned by the database when the operation failed.
publication_*n* (in)	publication	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. 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.

Remarks

If a procedure of this name exists, it is invoked when a database error is detected during the download phase of synchronization. The procedure is only invoked for errors where it is possible to ignore the error and continue with synchronization. For fatal errors, the sp_hook_dbmlsync_download_fatal_SQL_error procedure is called.

Caution
When continue is set to TRUE, dbmlsync simply ignores the database error and continues with synchronization. There is no attempt to retry the operation that failed. As a result, some or all of the download may be lost. The amount of data lost depends on the type of error encountered, when it occurred, and what steps the hook took to recover. It is very difficult to predict which data is lost and so this feature must be used with extreme caution. Most users would be best advised to not attempt to continue after a SQL error.

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 237
- "sp_hook_dbmlsync_download_fatal_sql_error (deprecated)" on page 261

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	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. 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.

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 237

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_name"        VARCHAR(128) NOT NULL ,
   "ml_user"             VARCHAR(128) NULL ,
   "event_time"          TIMESTAMP NULL,
   "table_name"          VARCHAR(128) NULL ,
   "upsert_count"        VARCHAR(128) NULL ,
   "delete_count"        VARCHAR(128) NULL ,
   "exit_code"           INTEGER NULL ,
   "status_retval"       VARCHAR(128) NULL ,
   "pubs"                 VARCHAR(128) NULL ,
   "sync_descr "          VARCHAR(128) NULL ,
    PRIMARY KEY ("event_id"),
);
```

The following example logs the MobiLink user, table name, and current timestamp immediately before a table is downloaded.

```
CREATE PROCEDURE sp_hook_dbmlsync_download_table_begin()
BEGIN
```

```
      DECLARE tbl VARCHAR(255);

      -- load the table name from #hook_dict
    SELECT #hook_dict.value
     INTO tbl
     FROM #hook_dict
     WHERE #hook_dict.name = 'table name';

    INSERT INTO SyncLog (event_name, ml_user, table_name
      ,event_time)
     SELECT 'download_table_begin', #hook_dict.value, tbl
      ,CURRENT TIMESTAMP
     FROM #hook_dict
     WHERE name = 'MobiLink user' ;
END;
```

sp_hook_dbmlsync_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.
publication_*n* (in)	publication	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. 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.

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 237

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_name"       VARCHAR(128) NOT NULL ,
   "ml_user"            VARCHAR(128) NULL ,
   "event_time"         TIMESTAMP NULL,
   "table_name"         VARCHAR(128) NULL ,
   "upsert_count"       VARCHAR(128) NULL ,
   "delete_count"       VARCHAR(128) NULL ,
   "exit_code"          INTEGER NULL ,
   "status_retval"      VARCHAR(128) NULL ,
   "pubs"                VARCHAR(128) NULL ,
   "sync_descr "         VARCHAR(128) NULL ,
    PRIMARY KEY ("event_id"),
);
```

The following example logs the MobiLink user, the table name and the number of inserted or updated rows immediately after a table is downloaded.

```
CREATE PROCEDURE sp_hook_dbmlsync_download_table_end()
BEGIN
    -- declare variables
    DECLARE tbl VARCHAR(255);
    DECLARE upsertCnt VARCHAR(255);
    DECLARE deleteCnt VARCHAR(255);

    -- load the table name from #hook_dict
    SELECT #hook_dict.value
     INTO tbl
     FROM #hook_dict
     WHERE #hook_dict.name = 'table name';

    -- load the upsert count from #hook_dict
    SELECT #hook_dict.value
     INTO upsertCnt
     FROM #hook_dict
     WHERE #hook_dict.name = 'upsert count';

    -- load the delete count from #hook_dict
    SELECT #hook_dict.value
     INTO deleteCnt
     FROM #hook_dict
     WHERE #hook_dict.name = 'delete count';

    INSERT INTO SyncLog (event_name, ml_user, table_name,
      upsert_count, delete_count, event_time)
     SELECT 'download_table_end', #hook_dict.value, tbl,
      upsertCnt, deleteCnt, CURRENT TIMESTAMP
     FROM #hook_dict
     WHERE name = 'MobiLink user' ;
END;
```

sp_hook_dbmlsync_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**	If set to **sync**, then dbmlsync retries the synchronization it just completed. The value **sync** replaces **true**, which is identical but is deprecated. If set to **none** (the default), then dbmlsync shuts down or restarts according to its command line arguments. The value **none** replaces **false**, which is identical but is deprecated. If set to **download** and the restartable download parameter is **true**, then dbmlsync attempts to restart the download that just failed.
exit code (in)	number	If set to anything other than zero (the default), this represents a synchronization error.
publication_*n* (in)	publication	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* 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**	Specifies the status returned by the MobiLink server when dbmlsync attempted to verify receipt of the upload. The status can be: ● **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, which would happen in cases such as download-only synchronization, a restarted download, or 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.
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 sp_hook_dbmlsync_all_error, sp_hook_dbmlsync_communication_error, sp_hook_dbmlsync_misc_error, or sp_hook_dbmlsync_sql_error.

Remarks

If a procedure of this name exists, it is called at the end of each synchronization.

Actions of this procedure are committed immediately after execution.

If an sp_hook_dbmlsync_end hook is defined so that the hook always sets the restart parameter to **sync**, and you specify multiple publications on the dbmlsync command line in the form -n pub1, -n pub2, and so on, then dbmlsync repeatedly synchronizes the first publication and never synchronizes the second.

See also

- "Introduction to dbmlsync hooks" on page 237
- "Synchronization event hook sequence" on page 237
- "Resuming failed downloads" [*MobiLink - Server Administration*]
- "Handling errors and warnings in event hook procedures" on page 241

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_dbmlsync_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_dbmlsync_log_rescan

Use this stored procedure to programmatically decide when a rescan is required.

Rows in #hook_dict table

Name	Value	Description
publication_*n* (in)	publication	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* 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.

Remarks

When more than one -n 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 199

Examples

The following example sets the rescan field in the #hook_dict table to TRUE 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_dbmlsync_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_*n* (in)	number	The log offset value where scanning is to begin. There is one value for each publication being uploaded. The numbering of *n* starts at zero. This value matches the Publication-*n*. For example, log offset_0 is the offset for publication_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 dbmlsync have different information about where the scanning should begin.
publication_*n* (in)	publication	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. 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.

Remarks

If a procedure of this name exists, it is called immediately before dbmlsync scans the transaction log to assemble the upload.

Actions of this procedure are committed immediately after execution.

See also

- “Synchronization event hook sequence” on page 237

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_name"       VARCHAR(128) NOT NULL ,
   "ml_user"            VARCHAR(128) NULL ,
   "event_time"         TIMESTAMP NULL,
   "table_name"         VARCHAR(128) NULL ,
   "upsert_count"       VARCHAR(128) NULL ,
   "delete_count"       VARCHAR(128) NULL ,
   "exit_code"          INTEGER NULL ,
   "status_retval"      VARCHAR(128) NULL ,
   "pubs"                VARCHAR(128) NULL ,
   "sync_descr "         VARCHAR(128) NULL ,
    PRIMARY KEY ("event_id"),
);
```

The following example logs the MobiLink user and current timestamp immediately before the transaction log is scanned for upload.

```
CREATE PROCEDURE sp_hook_dbmlsync_logscan_begin ()
BEGIN
 -- log the synchronization event
 INSERT INTO SyncLog (event_name, ml_user,event_time)
  SELECT 'logscan_begin', #hook_dict.value, CURRENT TIMESTAMP
  FROM #hook_dict
  WHERE name = 'MobiLink user' ;
END;
```

sp_hook_dbmlsync_logscan_end

Use this stored procedure to add custom actions immediately after the transaction log is scanned for upload.

Rows in #hook_dict table

Name	Value	Description
ending log offset (in)	number	The log offset value where scanning ended.
starting log offset_*n* (in)	number	The initial progress value for each subscription you synchronize. The *n* values correspond to those in Publication_*n*. 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_*n* (in)	publication	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. 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.

Remarks

If a procedure of this name exists, it is called immediately after dbmlsync has scanned the transaction log to assemble the upload.

Actions of this procedure are committed immediately after execution.

See also

- “Synchronization event hook sequence” on page 237

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_name"        VARCHAR(128) NOT NULL ,
   "ml_user"           VARCHAR(128) NULL ,
   "event_time"        TIMESTAMP NULL,
   "table_name"        VARCHAR(128) NULL ,
   "upsert_count"      VARCHAR(128) NULL ,
   "delete_count"      VARCHAR(128) NULL ,
   "exit_code"         INTEGER NULL ,
   "status_retval"     VARCHAR(128) NULL ,
   "pubs"               VARCHAR(128) NULL ,
   "sync_descr "        VARCHAR(128) NULL ,
    PRIMARY KEY ("event_id"),
);
```

The following example logs the MobiLink user and current timestamp immediately after the transaction log is scanned for upload. The #hook_dict log scan retry parameter indicates if the transaction log is scanned more than one time.

```
CREATE PROCEDURE sp_hook_dbmlsync_logscan_end ()
BEGIN

 DECLARE scan_retry VARCHAR(128);

 -- load the scan retry parameter from #hook_dict
 SELECT #hook_dict.value
  INTO scan_retry
  FROM #hook_dict
  WHERE #hook_dict.name = 'log scan retry';

 -- Determine if the log is being rescanned
 -- and log the synchronization event

 IF scan_retry='true' THEN
   INSERT INTO SyncLog (event_name, ml_user,event_time,sync_descr)
    SELECT 'logscan_end', #hook_dict.value, CURRENT TIMESTAMP,
     'Transaction log rescanned'
    FROM #hook_dict
    WHERE name = 'MobiLink user' ;
 ELSE
   INSERT INTO SyncLog (event_name, ml_user,event_time,sync_descr)
    SELECT 'logscan_end', #hook_dict.value, CURRENT TIMESTAMP,
     'Transaction log scanned normally'
    FROM #hook_dict
    WHERE name = 'MobiLink user' ;
 END IF;
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	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. 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.
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 the hook changes the value of the row, the new value is used in the next hook call. When you use this hook to pass state information to the sp_hook_dbmlsync_end hook, you can cause the _end hook to perform actions such as retrying the synchronization.

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* rows are set in the #hook_dict table.

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.

By default on Windows Mobile devices, synchronization tables are locked in exclusive mode, which means that this hook cannot successfully execute if it requires access to any of the synchronization tables. It also cannot execute if it needs to access synchronization tables and you set the dbmlsync extended option LockTables to EXCLUSIVE. See "LockTables (lt) extended option" on page 203.

Actions of this procedure are committed immediately after execution.

See also

- "Handling errors and warnings in event hook procedures" on page 241
- "sp_hook_dbmlsync_communication_error" on page 250
- "sp_hook_dbmlsync_all_error" on page 245
- "sp_hook_dbmlsync_sql_error" on page 297

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 publication.

```
dbmlsync -c eng=custdb;uid=DBA;pwd=sql -n test
```

Now, the error_log table contains the following row, associating the error with the error id 9931.

```
1,9931,
 'There is no synchronization subscription to publication "test"'
```

To provide custom error handling, check for the error id 9931 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 = 9931 THEN
  // handle invalid publication
 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_*n* (in)	publication	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. 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.
connection address (in|out)	connection address	When the hook is invoked, this is the address used in the most recent failed communication attempt. 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 communication attempt. For a list of protocol options, see "MobiLink client network protocol option summary" on page 35.
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 communication attempt. 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 communication attempt. For a list of network protocols, see "CommunicationType (ctp) extended option" on page 189.
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 dbmlsync was started with the -ra option. You can use this row to read or change the -ra option for the current synchronization only. See "-r option" on page 170.

Name	Value	Description
allow remote behind (in|out)	**true** | **false**	This is true only if dbmlsync was started with the -rb option. You can use this row to read or change the -rb option for the current synchronization only. See "-r option" on page 170.
retry (in|out)	**true** | **false**	Set this value to true if you want to retry a failed connection attempt. The default is FALSE.

Remarks

If a procedure of this name exists, it is called if dbmlsync 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, dbmlsync 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_dbmlsync_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 value
      INTO buf
      FROM #hook_dict
      WHERE name = 'user data';

  IF idx <= 5 THEN
        UPDATE #hook_dict
        SET value = idx
        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 value
  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
        WHERE name = 'user data';

       UPDATE #hook_dict
        SET value = 'TRUE'
        WHERE name = 'retry';
 END IF;
END;
```

sp_hook_dbmlsync_process_exit_code

Use this stored procedure to manage exit codes.

Rows in #hook_dict table

Name	Value	Description
publication_*n* (in)	publication	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* 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 dbmlsync to terminate.
aborted synchronization (in)	**true** | **false**	True when the hook is called because of an abort request from the sp_hook_dbmlsync_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_dbmlsync_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 dbmlsync 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.

Remarks

A dbmlsync session can run multiple synchronizations when you specify the -n 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* 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 237
- “sp_hook_dbmlsync_abort” on page 243

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_*n* (in)	publication	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* 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.

Remarks

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 dbmlsync extended option LockTables to off (LockTables is on by default).

During any synchronization where the upload was applied successfully and acknowledged by MobiLink, this hook is called after the sp_hook_dbmlsync_download_end hook and before the sp_hook_dbmlsync_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.

See also

- “Schema changes in remote clients” on page 79

Examples

The following example uses the sp_hook_dbmlsync_schema_upgrade procedure to add a column to the Dealer table on the remote database. If the upgrade is successful the sp_hook_dbmlsync_schema_upgrade hook is dropped.

```
CREATE PROCEDURE sp_hook_dbmlsync_schema_upgrade()
BEGIN
 -- Upgrade the schema of the Dealer table. Add a column:
 ALTER TABLE Dealer
  ADD dealer_description VARCHAR(128);

 -- If the schema upgrade is successful, drop this hook:
 UPDATE #hook_dict
  SET value = 'on success'
  WHERE name = 'drop hook';
END;
```

sp_hook_dbmlsync_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	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
extended options (out)	*opt*=*val*;...	Extended options to add for the next synchronization.

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 publication and MobiLink user entries, and they apply only until the next time the hook is called for the same synchronization.

Scheduling options may not be specified using this hook.

Actions of this procedure are committed immediately after execution.

See also

- "Synchronization event hook sequence" on page 237
- "MobiLink SQL Anywhere client extended options" on page 183
- "Priority order" on page 185

Examples

The following example uses sp_hook_dbmlsync_set_extended_options to specify the SendColumnNames extended option. The extended option is only applied if pub1 is synchronizing.

```
CREATE PROCEDURE sp_hook_dbmlsync_set_extended_options ()
BEGIN
 IF exists(SELECT * FROM #hook_dict
  WHERE name LIKE 'publication_%' AND value='pub1')
 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 options.

Name	Value	Description
publication_*n* (in)	publication	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. 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.
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 35.

Remarks

You can use this hook to set the network protocol and network protocol options.

The protocol and options can also be set in the sp_hook_dbmlsync_set_extended_options, a hook that is called at the beginning of a synchronization. sp_hook_dbmlsync_set_ml_connect_info is called immediately before dbmlsync 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_dbmlsync_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 dbmlsync hooks" on page 237
- "Synchronization event hook sequence" on page 237
- "CommunicationType (ctp) extended option" on page 189
- "MobiLink client network protocol option summary" on page 35
- "sp_hook_dbmlsync_set_extended_options" on page 292

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 is 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	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
start progress as timestamp_*n*	progress as timestamp	The starting progress for each publication being synchronized expressed as a timestamp, where *n* is the same integer used to identify the publication.
start progress as bigint_*n*	progress as bigint	The starting progress for each publication being synchronized expressed as a bigint, where *n* is the same integer used to identify the publication.
script version (n)	script version name	The MobiLink script version to be used for the synchronization.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
end progress is bigint (in|out)	TRUE | FALSE	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'). 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'). The default is FALSE.

Name	Value	Description
end progress (in|out)	timestamp	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. 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 time-stamp.

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 23:59:59:9999, which is 255,611,203,259,999.

See also

- "Custom progress values in scripted upload" on page 384
- "Synchronization event hook sequence" on page 237
- "Scripted upload" on page 373

sp_hook_dbmlsync_sql_error

Use this stored procedure to handle database errors that occur during synchronization. For example, you can implement the sp_hook_dbmlsync_sql_error hook to perform a specific action when a specific SQL error occurs.

Rows in #hook_dict table

Name	Value	Description
publication_*n* (in)	publication	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. 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)	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 the hook changes the value of the row, the new value is used in the next hook call. When you use this hook to pass state information to the sp_hook_dbmlsync_end hook, you can cause the _end hook to perform actions such as retrying the synchronization.
SQL code (in)	SQL error code	The SQL error code returned by the database when the operation failed. These values are defined in *sqlerr.h* in the *SDK\Include* subdirectory of your SQL Anywhere 11 installation.

Name	Value	Description
SQL state (in)	SQLSTATE value	The SQL state returned by the database when the operation failed.

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* 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*].

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.

By default on Windows Mobile devices, synchronization tables are locked in exclusive mode, which means that this hook cannot successfully execute if it requires access to any of the synchronization tables. It also cannot execute if it needs to access synchronization tables and you set the dbmlsync extended option LockTables to EXCLUSIVE. See "LockTables (lt) extended option" on page 203.

Actions of this procedure are committed immediately after execution.

See also

- "Handling errors and warnings in event hook procedures" on page 241
- "sp_hook_dbmlsync_all_error" on page 245
- "sp_hook_dbmlsync_communication_error" on page 250
- "sp_hook_dbmlsync_misc_error" on page 282
- "SQL Anywhere error messages" [*Error Messages*]

sp_hook_dbmlsync_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_*n* (in)	publication	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. 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.

Remarks

If a procedure of this name exists, it is called immediately before dbmlsync sends the upload.

Actions of this procedure are committed immediately after execution.

See also

- "Synchronization event hook sequence" on page 237

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_name"       VARCHAR(128) NOT NULL ,
   "ml_user"            VARCHAR(128) NULL ,
   "event_time"         TIMESTAMP NULL,
   "table_name"         VARCHAR(128) NULL ,
   "upsert_count"       VARCHAR(128) NULL ,
   "delete_count"       VARCHAR(128) NULL ,
   "exit_code"          INTEGER NULL ,
   "status_retval"      VARCHAR(128) NULL ,
   "pubs"                VARCHAR(128) NULL ,
   "sync_descr "         VARCHAR(128) NULL ,
    PRIMARY KEY ("event_id"),
);
```

The following example logs the MobiLink user and current timestamp immediately before the transmission of the upload.

```
CREATE PROCEDURE sp_hook_dbmlsync_upload_begin ()
BEGIN
 INSERT INTO SyncLog (event_name, ml_user,event_time)
  SELECT 'upload_begin', #hook_dict.value, CURRENT TIMESTAMP
```

```
    FROM #hook_dict
    WHERE name = 'MobiLink user';
  END;
```

sp_hook_dbmlsync_upload_end

Use this stored procedure to add custom actions after dbmlsync 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 Description.
upload status (in)	**retry** | **committed** | **failed** | **unknown**	Specifies the status returned by the MobiLink server when dbmlsync attempted to verify receipt of the upload. **retry** - The MobiLink server and dbmlsync had different values for the log offset from which the upload should start. The upload was not committed by the MobiLink server. The dbmlsync utility attempts to send another upload starting from a new log offset. **committed** - The upload was received by the MobiLink server and committed. **failed** - The MobiLink server did not commit the upload. **unknown** - Dbmlsync was started with the -tu option, causing transaction-level uploads. For each transaction that is uploaded, the sp_hook_dbmlsync_upload_begin and sp_hook_dbmlsync_upload_end hooks are called and the upload status value is **unknown** - each time except the last one.
publication_*n* (in)	publication	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. 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.

Name	Value	Description
authentication value (in)	value	This value is generated by the authenticate_user, authenticate_user_hashed, or authenticate_parameters script on the server. The value is an empty string when the upload status is unknown or when the upload_end hook is called after an upload is resent because of a conflict between the log offsets stored in the remote and consolidated databases.

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.

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_ABORTED_UPLOAD** The upload failed due to an error that occurred on the remote. Typical causes of the failure include communication errors and out-of-memory conditions.
- **UPLD_ERR_COMMUNICATIONS_FAILURE** A communication error occurred.
- **UPLD_ERR_LOG_OFFSET_MISMATCH** The upload failed because of conflict between log offset stored on the remote and consolidated databases.
- **UPLD_ERR_GENERAL_FAILURE** The upload failed for an unknown reason.
- **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_USERID_ALREADY_IN_USE** The user ID was already in use.
- **UPLD_ERR_DOWNLOAD_NOT_AVAILABLE** The upload was committed on the consolidated but an error occurred that prevented MobiLink from generating a download.
- **UPLD_ERR_PROTOCOL_MISMATCH** dbmlsync received unexpected data from the MobiLink server.
- **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.

See also

- "Synchronization event hook sequence" on page 237

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_name"       VARCHAR(128) NOT NULL ,
   "ml_user"            VARCHAR(128) NULL ,
   "event_time"         TIMESTAMP NULL,
   "table_name"         VARCHAR(128) NULL ,
   "upsert_count"       VARCHAR(128) NULL ,
   "delete_count"       VARCHAR(128) NULL ,
   "exit_code"          INTEGER NULL ,
   "status_retval"      VARCHAR(128) NULL ,
   "pubs"                VARCHAR(128) NULL ,
   "sync_descr "         VARCHAR(128) NULL ,
     PRIMARY KEY ("event_id"),
);
```

The following example logs the MobiLink user and current timestamp after dbmlsync verifies that the MobiLink server has received the upload.

```
CREATE PROCEDURE sp_hook_dbmlsync_upload_end ()
BEGIN

 DECLARE status_return_value VARCHAR(255);

 -- store status_return_value
  SELECT #hook_dict.value
  INTO status_return_value
  FROM #hook_dict
  WHERE #hook_dict.name = 'upload status';

   INSERT INTO SyncLog (event_name, ml_user,
     status_retval, event_time)
   SELECT 'upload_end', #hook_dict.value,
     status_return_value, CURRENT TIMESTAMP
  FROM #hook_dict
  WHERE name = 'MobiLink user';
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_*n* (in)	publication	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The *n* in publication_*n* and generation number_*n* match. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
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_*n* (in)	number	The generation numbers from the download file. There is one file generation number_*n* for each publication_*n* entry. The *n* in publication_*n* and generation number_*n* match. The numbering of *n* starts at zero.
user data (in)	string	The string specified with the dbmlsync -be option when the download file was created.
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.

Name	Value	Description
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 was used when the download file was created. If -bg was used, the generation numbers on the remote database are updated from the download file and normal generation number checks are not performed.

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 and SYSPUBLICATION system views.

This hook is called when the -ba option is specified. It is called before the download file is applied to the remote database.

The actions of this hook are committed immediately after it completes.

See also

- "-be option" on page 140
- "-bg option" on page 141
- "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_dbmlsync_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;
```

Dbmlsync API

Contents

Introduction to the 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 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.

Architecture

When using the Dbmlsync API, the client application instantiates and calls methods in the DbmlsyncClient class. This class communicates using TCP/IP with a separate process, dbmlsync server, that actually performs the synchronization by connecting to the MobiLink server and the remote database. Status information generated by synchronizations is communicated back to the client application through the GetEvent method of the DbmlsyncClient class.

More than one client can share the same dbmlsync server. However, each dbmlsync server can only synchronized a single remote database and each remote database can have only one dbmlsync server synchronizing it.

The dbmlsync server performs synchronizations one at a time. If it receives a synchronization request while it is already performing a synchronization, it queues that request and satisfies it later.

Dbmlsync API interfaces

There are 2 versions of the Dbmlsync API, one for C++ and one for .NET.

The C++ version is implemented in the DLL dbmlsynccli11.dll. Clients wishing to use this version should include the header dbmlsynccli.hpp in their C++ code and link against the import library dbmlsynccli11.lib. They must then distribute the dbmlsynccli11.dll file with their application. See "Dbmlsync API for C++" on page 309.

The .NET version of the API is implemented in the DLL iAnywhere.MobiLink.Client.dll. .NET applications can use the interface by including a reference to the DLL. See "Dbmlsync API for .NET" on page 322.

Dbmlsync API for C++

This section describes the methods in the C++ implementation of the DbmlsyncClient class.

The sample below shows a typical application using the C++ version of the Dbmlsync API to perform a synchronization to receive output events. The sample omits error handling for clarity. It is always good indent practice to check the return value from each API call.

Dbmlsync C++ sample

```
#include <stdio.h>
#include "dbmlsynccli.h"


int main( void ) {
    DbmlsyncClient *client;
    DBSC_SyncHdl    syncHdl;
    DBSC_Event     *ev1;

    client = DbmlsyncClient::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 eng=remote;dbn=rem1;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 );
}
```

The DbmlsyncClient class public methods in the C++ version of the API are shown below.

InstantiateClient method

Call the InstantiateClient method to instantiate a DbmlsyncClient class.

Syntax

static DbmlsyncClient ***InstantiateClient( );**

Remarks

The pointer returned by this method can be used to call the remaining methods in the class. After you are finished with the instance returned by InstantiateClient you can destroy it by calling delete on the pointer.

Returns

Returns a pointer to the new instance that has been created.

Returns null if an error occurs.

Init method

Initializes an instance of the class.

Syntax

bool **Init( );**

Remarks

This method must be the first method called after a DbmlSyncClient class is instantiated using the InstantiateClient method.

Returns

Returns true if the class instance was successfully initialized.

Returns false if the class instance was not successfully initialized. When false is returned, you can call the GetErrorInfo method to get more information about the failure. Other methods cannot be called until you have successfully initialized the instance. See "GetErrorInfo method" on page 317.

Fini method

Frees all resources used by this instance of the class.

Syntax

bool **Fini**

Remarks

You must call the Fini method before you delete an instance of the class.

Before you call the Fini method, you must call the Disconnect method if the instance is connected to a server.

Returns

Returns true if the method was successful.

Returns false if the method was not successful. When false is returned, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 317.

StartServer method

This method first checks to see if there is a dbmlsync server listening on the specified port. If a server is present, the 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.

Note
In 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 *dbmlsync.exe*.
- *dbmlsync.exe* is in the Windows directory.

Syntax

bool **StartServer(** unsigned port, const char *cmdline, unsigned timeout, DBSC_Starttype *starttype **);**

Parameters

- **port** The TCP port to check for an existing dbmlsync server. If a new server is started, it is set to listen on this port.
- **cmdline** A valid command line for starting a dbmlsync server. The command line may contain only the following options which have the same meaning that they do for the dbmlsync 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[cnoprsut], -wc, -wh. See "dbmlsync syntax" on page 131.

  The -c option must be specified.
- **timeout** The maximum time in milliseconds to wait after a dbmlsync server is started for it to be ready to accept requests. Use DBSC_INFINITY to wait forever.
- **starttype** This is an 'out' parameter. If starttype is non-null on entry and StartServer returns true, then on exit the variable pointed to by starttype is set to one of the following values:
    - **DBSC_SS_STARTED** Indicates that a new dbmlsync server was started.
    - **DBSC_SS_ALREADY_RUNNING** Indicates that an existing dbmlsync server was found, so no new server was started.

Returns

Returns true if the server was already running or was successfully started.

Returns false if the server was not successfully started or the server did not begin processing requests before the timeout expired. When false is returned, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 317.

Connect method

Opens a connection with a dbmlsync server that is already running on this computer.

Syntax

bool **Connect(** const char *host, unsigned port, const char *uid, const char *pwd **);**

Remarks

The database user id and password passed in 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.

Parameters

- **host** Reserved. Use NULL.
- **port** The TCP port on which the server is listening. Use the same port value that you specified when the server was started using 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

Returns true if a connection to the server was established.

Returns false if a connection to the server could not be established. When false is returned, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 317.

Disconnect method

Breaks the connection with a dbmlsync server created using the Connect method.

Syntax

bool **Disconnect( )**

Remarks

You should always call Disconnect when you are finished with a connection.

Returns

Returns true if the connection to the server is broken successfully.

Returns false if the connection to the server could not be broken. When false is returned, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 317.

Ping method

Sends a ping request to the dbmlsync server to check if the connection is good and if the server is live and responding to requests.

Syntax

bool **Ping(** unsigned timeout **)**

Remarks

You must be connected to a server to call this method.

Parameters

- **timeout** The maximum number of milliseconds to wait for the server to respond to the ping request. Use DBSC_INFINITY to wait forever.

Returns

Returns true if a response was received from the server to the ping request.

Returns false if a response to the ping request was not received. When false is returned, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 317.

Sync method

Request that the dbmlsync server perform a synchronization.

Syntax

DBSC_SyncHdl **Sync(** const char *profile_name, const char *extra_opts **)**

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.

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. 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.

Returns

Returns a DBSC_SyncHdl value which uniquely identifies this synchronization request. The returned handle 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 to get more information about the failure. See "GetErrorInfo method" on page 317.

ShutdownServer method

Shuts down the dbmlsync server to which the client is connected.

Syntax

bool **ShutdownServer(** DBSC_ShutdownType how **)**

Remarks

The Shutdown method returns immediately but there may be some delay before the server actually shuts down.

You must still call Disconnect after calling ShutdownServer.

The WaitForServerShutdown method can be used to wait until the server actually shuts down. See "WaitForServerShutdown method" on page 315.

Parameters

- **how** Indicates how urgently the server should be shutdown. The following values are supported:
  - **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.
  - **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.

Returns

Returns true if a shutdown request was successfully sent to the server.

Returns false if a shutdown request could not be sent. When false is returned, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 317.

WaitForServerShutdown method

The WaitForServerShutdown method returns when the server has shutdown or when the timeout expires, whichever comes first.

Syntax

bool **WaitForServerShutdown(** unsigned timeout **)**

Remarks

WaitForServerShutdown can only be called after the ShutdownServer method is called. See "ShutdownServer method" on page 314.

Parameters

- **timeout** Indicates the maximum time in milliseconds to wait for the server to shutdown. Use DBSC_INFINITY to wait forever.

Returns

Returns true if the method returned because the server shutdown.

Returns false otherwise. When false is returned, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 317.

CancelSync method

Allows a client to cancel a synchronization request previously made using the Sync method.

Syntax

bool **CancelSync(** DBSC_SyncHdl hdl **)**

Remarks

Only synchronization requests waiting to be serviced can be canceled. The server does not cancel a synchronization once it has begun.

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.

Parameters

- **hdl** The synchronization handle returned by the Sync method when the synchronization was requested.

Returns

Returns true if the synchronization request was successfully canceled.

Returns false if the synchronization request was not canceled. When false is returned, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 317.

GetEvent method

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.

Syntax

DBSC_GetEventRet **GetEvent(** DBSC_Event **event, unsigned timeout **)**

Remarks

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 event that are generated for a synchronization can be controlled using properties. See "SetProperty method" on page 319.

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. See "DBSC_Event structure" on page 320 and "FreeEventInfo method" on page 316.
- **timeout** Specifies the maximum number of milliseconds to wait if no event is immediately available to return. Use DBSC_INFINITY to wait forever.

Returns

Returns one of the following:

- **DBSC_GETEVENT_OK** Indicates that an event was successfully retrieved.
- **DBSC_GETEVENT_TIMED_OUT** Indicates that the timeout expired without any event being available to return.
- **DBSC_GETEVENT_FAILED** Indicates that no event was returned because of an error condition. When DBSC_GETEVENT_FAILED is returned, you can call the GetErrorInfo method to find out more about why the method failed. See "GetErrorInfo method" on page 317.

FreeEventInfo method

Frees memory associated with a DBSC_Event structure returned by the GetEvent method.

Syntax

bool **FreeEventInfo(** DBSC_Event *event **)**

Remarks

FreeEventInfo must be called on each DBSC_Event structure returned by the GetEvent method.

Parameters

- **event** A pointer to the DBSC_Event structure to be freed.

Returns

Returns true if the memory was successfully freed.

Returns false if the memory could not be freed. When false is returned, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 317.

GetErrorInfo method

Use this method to retrieve additional information about the failure immediately after another method in the interface fails.

Syntax

const DBSC_ErrorInfo ***GetErrorInfo( )**

Returns

Returns a pointer to a DBSC_ErrorInfo structure that contains information about the failure. The DBSC_ErrorInfo structure is defined as follows:

```
typedef struct {
    DBSC_ErrorType type;
    const char  *str1;
    const char  *str2;
    long int  val1;
    long int  val2;
    DBSC_SyncHdl hdl1;
} DBSC_ErrorInfo;
```

The contents of this structure may be overwritten the next time any class method is called.

Remarks

The type field contains a value that indicates the reason for the failure. Currently, type may take the following values:

- **DBSC_ERR_OK** No error occurred.
- **DBSC_ERR_NOT_INITIALIZED** The class has not been initialized by calling the Init method.
- **DBSC_ERR_ALREADY_INITIALIZED** The Init method was called on a class that was already initialized.
- **DBSC_ERR_NOT_CONNECTED** No connection to a dbmlsync server is in place.
- **DBSC_ERR_CANT_RESOLVE_HOST** Cannot resolve host information.
- **DBSC_ERR_CONNECT_FAILED** Connection failed.
- **DBSC_ERR_INITIALIZING_TCP_LAYER** Error initializing TCP layer.

- **DBSC_ERR_ALREADY_CONNECTED** Connect method failed because a connection was already in place.
- **DBSC_ERR_PROTOCOL_ERROR** This is an internal error.
- **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.
- **DBSC_ERR_TIMED _OUT** The timeout expired while waiting for a response from the server.
- **DBSC_ERR_STILL_CONNECTED** Could not Fini the class because it is still connected to the server.
- **DBSC_ERR_SYNC_NOT_CANCELED** The server could not cancel the synchronization request, probably because the synchronization was already in progress.
- **DBSC_ERR_INVALID_VALUE** An invalid property value was passed to the SetProperty method.
- **DBSC_ERR_INVALID_PROP_NAME** The specified property name is not valid.
- **DBCS_ERR_VALUE_TOO_LONG** The property value is too long. Properties must be less than DBCS_MAX_PROPERTY_LEN bytes long.
- **DBSC_ERR_SERVER_SIDE_ERROR** There was an error on the server.

  str1 points to a string returned by the server which may provide more information about the error.
- **DBSC_ERR_CREATE_PROCESS_FAILED** Unable to start a new dbmlsync server.
- **DBSC_ERR_READ_FAILED** Error occurred while reading data from the dbmlsync server.
- **DBSC_ERR_WRITE_FAILED** Error occurred while sending data to the dbmlsync server.
- **DBSC_ERR_NO_SERVER_RESPONSE** Failed to receive a response from the server that is required to complete the requested action.
- **DBSC_ERR_UID_OR_PWD_TOO_LONG** The UID or PWD specified is too long.
- **DBSC_ERR_UID_OR_PWD_NOT_VALID** The UID or PWD specified is not valid.
- **DBSC_ERR_INVALID_PARAMETER** One of the parameters passed to the function was not valid.
- **DBSC_ERR_WAIT_FAILED** An error occurred while waiting for the server to shutdown.
- **DBSC_SHUTDOWN_NOT_CALLED** WaitForServerShutdown method was called without first calling the ShutdownServer method.
- **DBSC_ERR_NO_SYNC_ACK** A synchronization request was sent to the server but no acknowledgement was received. There is no way to tell 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" on page 316.

str1, str2, val1, val2 and hdl1 contain additional information about the failure, and their meaning depends on the type value. For most type values there is no useful information in any of these fields. The exceptions are noted in the above in the previous list.

SetProperty method

Allows properties to be set on a class instance that modify various aspects of its behavior.

Syntax

bool **SetProperty(** const char *name, const char *value **)**

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 *dbmlsync.exe* when the StartServer method is called. When this property is not set, *dbmlsync.exe* is found using the path environment variable. See "StartServer method" on page 311.

The following 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". See "GetEvent method" on page 316.

The following is a list of the available properties and the event type that each controls:

Property name	Event type(s) controlled	Default
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
enable sync start	DBSC_EVENTTYPE_SYNC_START	1
enable sync done	DBSC_EVENTTYPE_SYNC_DONE	1

Parameters

- **name** The name of the property to set. This must be one of the property names defined above.
- **value** The value to set the property to. The string specified must contain less than DBCS_MAX_PROPERTY_LEN bytes.

Returns

Returns true if the property was successfully set.

Returns false if the property was not successfully set. When false is returned, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 317.

GetProperty method

Retrieves the current value of a property.

Syntax

bool **GetProperty(** const char *name, char *value **)**

Parameters

- **name** The name of the property to retrieve the value of. For a list of valid property names, see "SetProperty method" on page 319.
- **value** A buffer of at least DBSC_MAX_PROPERTY_LEN bytes where the value of the property is stored.

Returns

Returns true if the property was successfully retrieved.

Returns false if the property could not be retrieved. When false is returned, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 317.

DBSC_Event structure

The DBSC_Event structure contains information about a synchronization that has been requested. The structure is defined as follows:

```
typedef struct {
    DBSC_SyncHdl hdl;
    DBSC_EventType type;
    const char    *str1;
    const char   *str2;
    long int  val1;
    long int  val2;
    void    *data;
} DBSC_Event;
```

The hdl field identifies the synchronization request for which the structure contains information. This value matches the handle returned by the Sync method.

The type field identifies the type of event being reported.

The remaining fields contain additional data, the nature of which depends on the value of the type field. The following is a list of the possible type values and the meaning of the remaining fields associated with each:

- **DBSC_EVENTTYPE_ERROR_MSG** An error was generated by the synchronization and str1 points to the text of the error.

- **DBSC_EVENTTYPE_WARNING_MSG** A warning was generated by the synchronization and str1 points to the text of the warning.
- **DBSC_EVENTTYPE_INFO_MSG** An information message was generated by the synchronization and str1 points to the text of the message.
- **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.
- **DBSC_EVENTTYPE_PROGRESS_TEXT** The text associated with the progress bar has been updated and the new value is pointed to by str1.
- **DBSC_EVENTTYPE_TITLE** The title for the synchronization window/control has changed and the new title is pointed to by str1.
- **DBSC_EVENTTYPE_SYNC_START** The synchronization has begun. There is no additional information associated with this event.
- **DBSC_EVENTTYPE_SYNC_DONE** The synchronization is complete and val1 contains the exit code from the synchronization. A 0 value indicates success. A non-zero value indicates that the synchronization failed.

Dbmlsync API for .NET

This section describes the methods in the .NET implementation of the DbmlsyncClient class.

The sample below shows a typical application using the .NET version of the Dbmlsync API to perform a synchronization to receive output events. The sample omits error handling for clarity. It is always good indent practice to check the return value from each API call.

Dbmlsync .Net sample

```
using System;
using System.Collections.Generic;
using System.Text;
using iAnywhere.MobiLink.Client;

namespace ConsoleApplication6
{
    class Program
    {
        static void Main(string[] args)
        {
            DbmlsyncClient cli1;
            DBSC_StartType st1;
            DBSC_Event ev1;
            UInt32 syncHdl;

            cli1 = DbmlsyncClient.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\\asa1100r\\bin32");

            cli1.StartServer(3426,
             "-c eng=cons;dbn=rem1;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();
            }
        }
    }
```

The DbmlsyncClient class public methods in the .NET version of the API are shown below.

InstantiateClient method

Call the InstantiateClient method to instantiate a DbmlsyncClient class.

Syntax

static DbmlsyncClient **InstantiateClient()**

Remarks

The object returned by this method can be used to call the remaining methods in the class.

Returns

Returns a new DbmlsyncClient instance that has been created. If an error occurs, null is returned.

Init method

Initializes an instance of the class.

Syntax

Boolean **Init()**

Remarks

This method must be the first method called after a class is instantiated using the InstantiateClient method.

Returns

Returns true if the class instance was successfully initialized.

Returns false if the class instance was not successfully initialized. When false is returned, you can call the GetErrorInfo method to get more information about the failure. Other methods cannot be called until you have successfully initialized the instance. See "GetErrorInfo method" on page 329.

StartServer method

This method first checks to see if there is a dbmlsync server listening on the specified port. If a server is present, the 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.

Note
In 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 *dbmlsync.exe*.
- *dbmlsync.exe* is in the Windows directory.

Syntax

Boolean **StartServer(**Int32 port, String cmdline, UInt32 timeout, out DBSC_Starttype starttype **)**

Parameters

- **port** The TCP port to check for an existing dbmlsync server. If a new server is started, it is set to listen on this port.
- **cmdline** A valid command line for starting a dbmlsync server. The command line may contain only the following options which have the same meaning that they do for the dbmlsync 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[cnoprsut], -wc, -wh. See "dbmlsync syntax" on page 131.

  The -c option must be specified.
- **timeout** The maximum time in milliseconds to wait after a dbmlsync server is started for it to be ready to accept requests. Use DBSC_INFINITY to wait forever. The DBSC_INFINITY constant is defined within the DbmlSyncClient class and not in the namespace, so you need to preface the constant. For example, `timeout = DbmlSyncClient.DBSC_INFINITY;`.
- **starttype** This is an 'out' parameter. If StartServer returns true then on exit, starttype is set to one of the following values:
  - **DBSC_SS_STARTED** Indicates that a new dbmlsync server was started.
  - **DBSC_SS_ALREADY_RUNNING** Indicates that an existing dbmlsync server was found, so no new server was started.

Returns

Returns true if the server was already running or was successfully started.

Returns false if the server was not successfully started or the server did not begin processing requests before the timeout expired. When false is returned, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 329.

Connect method

Opens a connection with a dbmlsync server that is already running on this computer.

Syntax

Boolean **Connect(**String host, Int32 port, String uid, String pwd**)**

Remarks

The database user id and password passed in are used to validate whether this client has enough permissions to synchronize the database. When synchronizations are performed, the user id specified with the -c option when the dbmlsync server started is used.

Parameters

- **host** Reserved. Use null.
- **port** The TCP port on which the server is listening. Use the same port value that you specified when the server was started using 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

Returns true if a connection to the server was established.

Returns false if a connection to the server could not be established. When false is returned, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 329.

Disconnect method

Breaks the connection with a dbmlsync server created using the Connect method.

Syntax

Boolean **Disconnect()**

Remarks

You should always call Disconnect when you are finished with a connection.

Returns

Returns true if the connection to the server is broken successfully.

Returns false if the connection to the server could not be broken. When false is returned, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 329.

Ping method

Sends a ping request to the dbmlsync server to check if the connection is good and if the server is live and responding to requests.

Syntax

Boolean **Ping(**UInt32 timeout**)**

Remarks

You must be connected to a server to call this method.

Parameters

- **timeout** The maximum number of milliseconds to wait for the server to respond to the ping request. Use DBSC_INFINITY to wait forever. The DBSC_INFINITY constant is defined within the DbmlSyncClient class and not in the namespace, so you need to preface the constant. For example, `timeout = DbmlSyncClient.DBSC_INFINITY;`.

Returns

Returns true if a response was received from the server to the ping request.

Returns false if a response to the ping request was not received. When false is returned, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 329.

Sync method

Request that the dbmlsync server perform a synchronization.

Syntax

UInt32 **Sync(** string profile_name, string extra_opts **)**

Remarks

You must be connected to the server before calling this method.

At least one of syncName and opts must be non-null.

Parameters

- **profile_name** 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 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. The options specified in the string 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.

Returns

Returns an integer value which uniquely identifies this synchronization request. The returned value is only valid until the client disconnects from the server.

Returns NULL_SYNCHDL if an error prevents the synchronization request from being created. When this happens, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 329.

ShutdownServer method

Shuts down the dbmlsync server to which the client is connected.

Syntax

Boolean **ShutdownServer(**DBSC_ShutdownType how**)**

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. See "WaitForServerShutdown method" on page 327.

Parameters

- **how** Indicates how urgently the server should be shutdown. The following values are supported:
  - **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.
  - **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.

Returns

Returns true if a shutdown request was successfully sent to the server.

Returns false if a shutdown request could not be sent. When false is returned, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 329.

WaitForServerShutdown method

The WaitForServerShutdown method returns when the server has shutdown or when the timeout expires, whichever comes first.

Syntax

Boolean **WaitForServerShutdown(** UInt32 timeout **)**

Remarks

WaitForServerShutdown can only be called after the ShutdownServer method is called. See "ShutdownServer method" on page 327.

Parameters

- **timeout** Indicates the maximum time in milliseconds to wait for the server to shutdown. Use DBSC_INFINITY to wait forever. The DBSC_INFINITY constant is defined within the DbmlSyncClient class and not in the namespace, so you need to preface the constant. For example, `timeout = DbmlSyncClient.DBSC_INFINITY;`.

Returns

Returns true if the method returned because the server shutdown.

Returns false otherwise. When false is returned, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 329.

CancelSync method

Allows a client to cancel a synchronization request previously made using the Sync method.

Syntax

Boolean **CancelSync(**UInt32 hdl**)**

Remarks

Only synchronization requests waiting to be serviced can be canceled. The server does not cancel a synchronization once it has begun.

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.

Parameters

- **hdl** The synchronization handle returned by the Sync method when the synchronization was requested.

Returns

Returns true if the synchronization request was successfully canceled.

Returns false if the synchronization request was not canceled. When false is returned, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 329.

GetEvent method

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.

Syntax

DBSC_GetEventRet **GetEvent(**out DBSC_Event ev, UInt32 timeout**)**

Remarks

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 event that are generated for a synchronization can be controlled using properties. See "SetProperty method" on page 332.

Parameters

- **ev** If the return value is DBSC_GETEVENT_OK then the event is filled in with information about the event that has been retrieved. See "DBSC_Event structure" on page 320.
- **timeout** Specifies the maximum number of milliseconds to wait if no event is immediately available to return. Use DBSC_INFINITY to wait forever. The DBSC_INFINITY constant is defined within the DbmlSyncClient class and not in the namespace, so you need to preface the constant. For example, `timeout = DbmlSyncClient.DBSC_INFINITY;`.

Returns

Returns one of the following:

Return value	Description
DBSC_GETEVENT_OK	Indicates that an event was successfully retrieved.
DBSC_GETEVENT_TIMED_OUT	Indicates that the timeout expired without any event being available to return.
DBSC_GETEVENT_FAILED	Indicates that no event was returned because of an error condition. When DBSC_GETEVENT_FAILED is returned, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 329.

GetErrorInfo method

Use this method to retrieve additional information about the failure immediately after another method in the interface fails.

Syntax

DBSC_ErrorInfo **GetErrorInfo()**

Returns

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. The DBSC_ErrorInfo structure is defined as follows:

```
public struct DBSC_ErrorInfo
 {
   public DBSC_ErrorType type;
```

```
        public String str1;
        public String str2;
        public Int32 val1;
        public Int32 val2;
        public UInt32 hdl1;
    }
```

Remarks

The type field contains a value that indicates the reason for the failure. Currently, type may take the following values:

Type value	Description
DBSC_ERR_OK	No error occurred.
DBSC_ERR_NOT_INITIALIZED	The class has not been initialized by calling the init method.
DBSC_ERR_ALREADY_INITIALIZED	The init method was called on a class that was already initialized.
DBSC_ERR_NOT_CONNECTED	No connection to a dbmlsync server is in place.
DBSC_ERR_CANT_RESOLVE_HOST	Cannot resolve host information.
DBSC_ERR_CONNECT_FAILED	Connection failed.
DBSC_ERR_INITIALIZING_TCP_LAYER	Error initializing TCP layer.
DBSC_ERR_ALREADY_CONNECTED	Connection failed because a connection was already in place.
DBSC_ERR_PROTOCOL_ERROR	This is an internal error.
DBSC_ERR_CONNECTION_REJECTED	The connection was rejected by the dbmlsync server.
DBSC_ERR_TIMED _OUT	The timeout expired while waiting for a response from the server.
DBSC_ERR_STILL_CONNECTED	Could not Fini the class because it is still connected to the server.
DBSC_ERR_SYNC_NOT_CANCELED	The server could not cancel the synchronization request, probably because the synchronization was already in progress.
DBSC_ERR_INVALID_VALUE	An invalid property value was passed to the SetProperty method.

Type value	Description
DBSC_ERR_INVALID_PROP_NAME	The specified property name is not valid.
DBCS_ERR_VALUE_TOO_LONG	The property value is too long. Properties must be less than DBCS_MAX_PROPERTY_LEN bytes long.
DBSC_ERR_SERVER_SIDE_ERROR	There was an error on the server.
DBSC_ERR_CREATE_PROC-ESS_FAILED	Unable to start a new dbmlsync server.
DBSC_ERR_READ_FAILED	Error occurred while reading data from the dbmlsync serv-er.
DBSC_ERR_WRITE_FAILED	Error occurred while sending data to the dbmlsync server.
DBSC_ERR_NO_SERVER_RESPONSE	Failed to receive a response from the server that is required to complete the requested action.
DBSC_ERR_UID_OR_PWD_TOO_LO NG	The UID or PWD specified is too long.
DBSC_ERR_UID_OR_PWD_NOT_VA LID	The UID or PWD specified is not valid.
DBSC_ERR_INVALID_PARAMETER	One of the parameters passed to the function was not valid.
DBSC_ERR_WAIT_FAILED	An error occurred while waiting for the server to shutdown.
DBSC_SHUTDOWN_NOT_CALLED	WaitForServerShutdown method was called without first calling the ShutdownServer method.
DBSC_ERR_NO_SYNC_ACK	A synchronization request was sent to the server but no ac-knowledgement was received. There is no way to tell if the server received the request.
DBSC_ERR_CONNECTION_REJEC-TED	str1 points to a string returned by the server which may provide more information about why the connection at-tempt was rejected.
DBSC_ERR_NO_SYNC_ACK	hdl1 is the handle for the sync request that was sent. If the server received the request, this handle can be used to iden-tify events for the synchronization retrieved using the Ge-tEvent method.
DBSC_ERR_SERVER_SIDE_ERROR	str1 points to a string returned by the server which may provide more information about the error.

str1, str2, val1, val2 and hdl1 contain additional information about the failure and their meaning depends on the type value. For most type values there is no useful information in any of these fields. The exceptions are:

- DBSC_ERR_CONNECTION_REJECTED
- DBSC_ERR_NO_SYNC_ACK
- DBSC_ERR_SERVER_SIDE_ERROR

SetProperty method

Allows properties to be set on a class instance that modify various aspects of its behavior.

Syntax

Boolean **SetProperty(**String name, String Value**)**

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 *dbmlsync.exe* when the StartServer method is called. When this property is not set, *dbmlsync.exe* is found using the path environment variable. See "StartServer method" on page 323.

The following properties control the types of events that are returned by the "GetEvent method" on page 328. 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 list of the available properties and the event type that each controls:

Property name	Event type(s) controlled	Default
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
enable sync start	DBSC_EVENTTYPE_SYNC_START	1
enable sync done	DBSC_EVENTTYPE_SYNC_DONE	1

Parameters

- **name** The name of the property to set. This must be one of the property names defined above.
- **value** The value to set the property to.

Returns

Returns true if the property was successfully set.

Returns false if the property was not successfully set. When false is returned, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 329.

GetProperty method

Retrieves the current value of a property.

Syntax

Boolean **GetProperty(**String name, out String Value**)**

Parameters

- **name** The name of the property to retrieve the value of. For a list of valid property names, see "SetProperty method" on page 332.
- **value** On exit, the value of the property is stored in this variable.

Returns

Returns true if the property was successfully retrieved.

Returns false if the property could not be retrieved. When false is returned, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 329.

Fini method

Frees all resources used by this instance of the class.

Syntax

Boolean **Fini()**

Remarks

You must call the Fini method before you delete an instance of the class.

Before you call the Fini method, you must call the Disconnect method if the instance is connected to a server.

Returns

Returns true if the method was successful.

Returns false if the method was not successful. When false is returned, you can call the GetErrorInfo method to get more information about the failure. See "GetErrorInfo method" on page 329.

DBSC_Event structure

The DBSC_Event structure contains information about a synchronization that has been requested. The structure is defined as follows:

```
public struct DBSC_Event
 {
   public UInt32 hdl;
   public DBSC_EventType type;
   public String str1;
   public String str2;
   public Int32 val1;
   public Int32 val2;
 }
```

The hdl field identifies the synchronization request for which the structure contains information. This value matches the value returned by the Sync method.

The type field identifies the type of event being reported.

The remaining fields contain additional data, the nature of which depends on the value of the type field. The following is a list of the possible type values and the meaning of the remaining fields associated with each:

Value	Description
DBSC_EVENTTYPE_ERROR_MSG	An error was generated by the synchronization and str1 contains the text of the error.
DBSC_EVENTTYPE_WARNING_MSG	A warning was generated by the synchronization and str1 contains the text of the warning.
DBSC_EVENTTYPE_INFO_MSG	An information message was generated by the synchronization and str1 contains the text of the message.
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.
DBSC_EVENTTYPE_PROGRESS_TEXT	The text associated with the progress bar has been updated and contains the new value.
DBSC_EVENTTYPE_TITLE	The title for the synchronization window/control has changed and str1 contains the new title.
DBSC_EVENTTYPE_SYNC_START	The synchronization has begun. There is no additional information associated with this event.

Value	Description
DBSC_EVEN-TTYPE_SYNC_DONE	The synchronization is complete and val1 contains the exit code from the synchronization. A 0 value indicates success. A non-zero value indicates that the synchronization failed.

Dbmlsync integration component (deprecated)

Contents

Note
The Dbmlsync integration component has been deprecated. In its place, use the dbmlsync programming interface. See "Dbmlsync API" on page 307.

Introduction to Dbmlsync integration component

Note
The Dbmlsync integration component has been deprecated. In its place, use the dbmlsync programming interface. See "Dbmlsync API" on page 307.

The Dbmlsync 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 Dbmlsync 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 dbmlsync 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 Dbmlsync 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 dbmlsync

As an alternative to the Dbmlsync integration component, you can use DBTools interface for dbmlsync.

See "Database tools interface" [*SQL Anywhere Server - Programming*].

Supported platforms

You can use the Dbmlsync 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 Dbmlsync integration component

The Dbmlsync 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.

Dbmlsync integration component methods

The following are methods implemented by the DbmlsyncCOM.Dbmlsync class.

Run method

Begins one or more synchronizations using dbmlsync command line options.

Syntax

Run(ByVal *cmdLine* As String **)**
Member of **DbmlsyncCOM.Dbmlsync**

Parameters

cmdLine	A string specifying dbmlsync options.

Remarks

For a list of options, see "dbmlsync syntax" on page 131.

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 dbmlsync command line utility. For example, the following command line and Run method invocation are equivalent:

```
dbmlsync -c uid=DBA;pwd=sql
```

```
dbmlsync1.Run "-c uid=DBA;pwd=sql"
```

Example

The following example initiates a synchronization for a remote database called remote1.

```
dbmlsync1.Run "-c eng=remote1;uid=DBA;pwd=sql"
```

Stop method

Requests active synchronizations to terminate.

Syntax

Stop( )
Member of **DbmlsyncCOM.Dbmlsync**

Remarks

The Stop method issues a request to terminate any active synchronizations. It returns immediately.

The stop button built into the visual Dbmlsync integration component automatically invokes this method.

Example

The following example stops synchronizations being run by the Dbmlsync integration component instance dbmlsync1.

```
dbmlsync1.Stop
```

Dbmlsync integration component properties

Dbmlsync integration component properties let you customize the behavior of the component and examine the state of a running synchronization.

Path property

Specifies the location of *dbmlsync.exe*.

Syntax

Public Property **Path( )** As String
Member of **DbmlsyncCOM.Dbmlsync**

Remarks

You do not need to set this property if *dbmlsync.exe* is located in a directory specified by the Windows PATH environment variable.

Example

The following example sets the path of a Dbmlsync integration component instance.

```
dbmlsync1.Path = "c:\program files\SQL Anywhere 11\bin32"
```

UploadEventsEnabled property

Enables the UploadRow event.

Syntax

Public Property **UploadEventsEnabled()** As Boolean
Member of **DbmlsyncCOM.Dbmlsync**

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" on page 359.

Example

The following example sets UploadEventsEnabled to true:

```
dbmlsync1.UploadEventsEnabled = True
```

DownloadEventsEnabled property

Enables the DownloadRow event.

Syntax

Public Property **DownloadEventsEnabled( )** As Boolean
Member of **DbmlsyncCOM.Dbmlsync**

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” on page 350.

Example

The following example sets DownloadEventsEnabled to true:

```
dbmlsync1.DownloadEventsEnabled = True
```

ErrorMessageEnabled property

Prevents the Message event from being called for messages of type MsgError.

Syntax

Public Property **ErrorMessageEnabled( )** As Boolean
Member of **DbmlsyncCOM.Dbmlsync**

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” on page 354.

Example

The following example sets ErrorMessageEnabled to false:

```
dbmlsync1.ErrorMessageEnabled = False
```

WarningMessageEnabled property

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” on page 354.

Example

The following example sets WarningMessageEnabled to false:

```
dbmlsync1.WarningMessageEnabled = False
```

InfoMessageEnabled property

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” on page 354.

Example

The following example sets InfoMessageEnabled to false:

```
dbmlsync1.InfoMessageEnabled = False
```

DetailedInfoMessageEnabled property

Prevents the Message event from being called for messages of type MsgDetailedInfo.

Syntax

Public Property **DetailedInfoMessageEnabled( )** As Boolean
Member of **DbmlsyncCOM.Dbmlsync**

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” on page 354.

Example

The following example sets DetailedInfoMessageEnabled to false:

```
dbmlsync1.DetailedInfoMessageEnabled = False
```

UseVB6Types property

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 **DbmlsyncCOM.Dbmlsync**

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" on page 361
- "UploadRow event" on page 359
- "DownloadRow event" on page 350

Example

The following example enables data type coercion for a Dbmlsync integration component instance used in Visual Basic 6.0:

```
dbmlsync1.UseVB6Types = True
```

ExitCode property

Returns the exit code from synchronizations started by the most recent Run method invocation.

Syntax

Public Property **ExitCode( )** As Integer
Member of **DbmlsyncCOM.Dbmlsync**

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 dbmlsync1_DoneExecution() Handles dbmlsync1.DoneExecution
            MsgBox(dbmlsync1.ExitCode)
        End Sub
```

EventChannelSize property

Specifies the size of an internal buffer used for processing method calls.

Syntax

Public Property **EventChannelSize( )** As Integer
Member of **DbmlsyncCOM.Dbmlsync**

Remarks

Most users never have to change this property.

DispatchChannelSize property

Specifies the size of an internal buffer used for processing event information.

Syntax

Public Property **DispatchChannelSize( )** As Integer
Member of **DbmlsyncCOM.Dbmlsync**

Remarks

Most users never have to change this property.

Dbmlsync integration component events

Events provide a mechanism for client applications to receive and act on information about the progress of a synchronization.

BeginDownload event

The BeginDownload event is triggered at the beginning of the download stage of a synchronization.

Syntax

Public Event **BeginDownload( )**
Member of **DbmlsyncCOM.Dbmlsync**

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 dbmlsync1_BeginDownload()
Handles dbmlsync1.BeginDownload

        MsgBox("Beginning Download")

End Sub
```

BeginLogScan event

The BeginLogScan event is triggered immediately before dbmlsync 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 **DbmlsyncCOM.Dbmlsync**

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 dbmlsync 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 dbmlsync1_BeginLogScan(
 ByVal rescanLog As Boolean
)
Handles dbmlsync1.BeginLogScan

        MsgBox("Begin Log Scan")

End Sub
```

BeginSynchronization event

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

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 dbmlsync1_BeginUpload()
Handles dbmlsync1.BeginUpload

        MsgBox("Begin Upload")

End Sub
```

ConnectMobilink event

The ConnectMobilink event is triggered immediately before the component connects to the MobiLink server.

Syntax

Public Event **ConnectMobilink( )**
Member of **DbmlsyncCOM.Dbmlsync**

Remarks

Use this event to add custom actions immediately before the remote database connects to the MobiLink server. At this stage, dbmlsync 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 dbmlsync1_ConnectMobilink()
Handles dbmlsync1.ConnectMobilink

        MsgBox("Connecting to the MobiLink server")

End Sub
```

DisconnectMobilink event

The DisconnectMobilink event is triggered immediately after the component disconnects from the MobiLink server.

Syntax

Public Event **DisconnectMobilink( )**
Member of **DbmlsyncCOM.Dbmlsync**

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 dbmlsync1_DisconnectMobilink()
Handles dbmlsync1.DisconnectMobilink

        MsgBox("Disconnected from the MobiLink server")

End Sub
```

DoneExecution event

The DoneExecution event is triggered when all synchronizations started by a Run method invocation have completed.

Syntax

Public Event **DoneExecution( )**
Member of **DbmlsyncCOM.Dbmlsync**

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 dbmlsync1_DoneExecution()
Handles dbmlsync1.DoneExecution

        MsgBox(dbmlsync1.ExitCode)

End Sub
```

DownloadRow event

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" on page 361.

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" on page 342.

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

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 **DbmlsyncCOM.Dbmlsync**

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 dbmlsync1_EndDownload(
 ByVal upsertRows As Integer,
 ByVal deleteRows As Integer
)
Handles dbmlsync1.EndDownload

    MsgBox("Download complete." + _
     CStr(upsertRows) + "Rows updated or inserted" + _
     CStr(deleteRows) + "Rows deleted")

End Sub
```

EndLogScan event

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

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 dbmlsync1_EndSynchronization(
 ByVal ExitCode As Integer,
 ByRef restart As Boolean
)
Handles dbmlsync1.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

The EndUpload event is triggered immediately after transmission of the upload to the MobiLink server.

Syntax

Public Event **EndUpload( )**
Member of **DbmlsyncCOM.Dbmlsync**

Remarks

Use this event to add custom actions immediately after transmission of the upload from dbmlsync to the MobiLink server.

Example

The following Visual Basic .NET example outputs a message when the EndUpload event is triggered.

```
Private Sub dbmlsync1_EndUpload()
Handles dbmlsync1.EndUpload

    MsgBox("End Upload")

End Sub
```

Message event

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 DbmlsyncCOM.MessageClass.MsgInfo
            lstMessages.Items.Add("Info: " + msg)
        Case DbmlsyncCOM.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 dbmlsync1_Message(ByVal msgClass As DbmlsyncCOM.MessageClass,
ByVal msgId As Integer, ByVal msg As String) Handles dbmlsync1.Message
  If msgClass = DbmlsyncCOM.MessageClass.MsgError Then
   lstMessages.Items.Add("Error: " + msgId.ToString() + " " + msg)
  End If
End Sub
```

To see possible error id values, test run the Dbmlsync integration component. For example, if dbmlsync 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 Dbmlsync 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 dbmlsync1_Message
```

```
(
 ByVal msgClass As DbmlsyncCOM.MessageClass,
 ByVal msgId As Integer, ByVal msg As String ) Handles dbmlsync1.Message

 If msgClass = DbmlsyncCOM.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 dbmlsync1_EndSynchronization(ByVal ExitCode As Integer, _
    ByRef restart As Boolean _
    ) Handles dbmlsync1.EndSynchronization


    If restartSynchronization = 1 Then
      If numberOfRestarts < 5 Then
         restart = True
         numberOfRestarts = numberOfRestarts + 1
      End If
    End If
End Sub
```

ProgressIndex event

The ProgressIndex event is triggered when dbmlsync updates its progress bar.

Syntax

Public Event **ProgressIndex(_**
  ByVal *index* As Integer, _
  ByVal *max* As Integer _
)
Member of **DbmlsyncCOM.Dbmlsync**

Parameters

index An integer representing the progress of the synchronization.

max The maximum progress value. The percentage done = `index/max x 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 dbmlsync1_ProgressIndex(
 ByVal index As Integer,
 ByVal max As Integer
)
Handles dbmlsync1.ProgressIndex

    If max <> 0 Then
        ProgressBar1.Maximum = max
    End If
    ProgressBar1.Value = index

End Sub
```

ProgressMessage event

The ProgressMessage event is triggered when synchronization progress information changes.

Syntax

Public Event **ProgressMessage(** ByVal *msg* As String **)**
Member of **DbmlsyncCOM.Dbmlsync**

Parameters

msg	The new progress string.

Remarks

Use this event to receive the string normally displayed with the dbmlsync progress bar.

Example

The following Visual Basic .NET example sets the value of a progress label when the ProgressMessage event is triggered.

```
Private Sub dbmlsync1_ProgressMessage(
 ByVal msg As String
)
Handles dbmlsync1.ProgressMessage

    lblProgressMessage.Text = msg

End Sub
```

SetTitle event

The SetTitle event is triggered when status information changes. In the dbmlsync utility, this information is displayed in the title bar.

Syntax

Public Event **SetTitle(** ByVal *title* **)** As String
)
Member of **DbmlsyncCOM.Dbmlsync**

Parameters

title	The title in the dbmlsync window title bar.

Remarks

Use this event to receive the title normally seen on the dbmlsync 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 dbmlsync1_SetTitle(
 ByVal title As String
)
Handles dbmlsync1.SetTitle

        Me.Text = title

End Sub
```

UploadAck event

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 DbmlsyncCOM.UploadAckStatus _
)
Member of **DbmlsyncCOM.Dbmlsync**

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 dbmlsync 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 dbmlsync1_UploadAck(ByVal status As DbmlsyncCOM.UploadAckStatus)
Handles dbmlsync1.UploadAck
```

```
    If status = DbmlsyncCOM.UploadAckStatus.StatFailed Then
        MsgBox("Upload Failed")
    End If

End Sub
```

UploadRow event

The UploadRow event is triggered when a row is uploaded to the MobiLink server.

Syntax

Public Event **UploadRow(**
 ByVal *rowData* As DbmlsyncCOM.IRowTransferData
)
Member of **DbmlsyncCOM.Dbmlsync**

Parameters

rowData An IRowTransferData object containing details about the uploaded row.

See "IRowTransferData interface" on page 361.

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" on page 342.

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 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
```

WaitingForUploadAck event

The WaitingForUploadAck event is triggered when the component begins waiting for upload acknowledgement from the MobiLink server.

Syntax

Public Event **WaitingForUploadAck( )**
Member of **DbmlsyncCOM.Dbmlsync**

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 dbmlsync1_WaitingForUploadAck()
Handles dbmlsync1.WaitingForUploadAck

    MsgBox("Waiting for Upload Acknowledgement")

End Sub
```

IRowTransferData interface

Public Interface **IRowTransferData**
Member of **DbmlsyncCOM**

The UploadRow and DownloadRow events accept DbmlsyncCOM.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

Specifies the operation performed on the row.

Syntax

Public Property **RowOperation( )** As DbmlsyncCOM.RowEventOp
Member of **DbmlsyncCOM.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

The name of the table on which an upload or download operation occurred.

Syntax

Public Property **TableName( )** As String
Member of **DbmlsyncCOM.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” on page 359.

Example

The following is a Visual Basic .NET example.

```
Private Sub dbmlsync1_UploadRow(
 ByVal rowData As DbmlsyncCOM.IRowTransferData
)
Handles dbmlsync1.UploadRow

    MsgBox ("Table name:" + rowData.TableName)

End Sub
```

ColumnName property

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 **DbmlsyncCOM.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" on page 363.

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" on page 359.

```
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
```

ColumnValue property

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" on page 363.

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" on page 359.

```
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

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" on page 359.

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

Contents

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" [*SQL Anywhere Server - Programming*].

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 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 receiving synchronization and progress information.

  See "a_sync_db structure" [*SQL Anywhere Server - Programming*].

- **a_syncpub structure** This structure holds publication information. You can specify a linked list of publications 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*].

Dbmlsync Integration Component

As an alternative to the DBTools interface for dbmlsync, you can use the dbmlsync API.

See "Dbmlsync API" on page 307.

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 "Introduction to the database tools interface" [*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 option" on page 142.

- 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. In most cases this is the value you should return.
    - **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.msgqueuertn = 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 publications should be synchronized.

   Each node in the linked list corresponds to one instance of the -n option on the dbmlsync command line.

   - Declare an a_syncpub instance. For example, call it publication_info:

     ```
     a_syncpub publication_info;
     ```

   - Initialize a_syncpub fields, specifying publications you want to synchronize.

     For example, to identify the template_p1 and template_p2 publications together in a single synchronization session:

     ```
     publication_info.next = NULL; // linked list terminates
     publication_info.pub_name = "template_p1,template_p2";
     publication_info.ext_opt  = "sv=template_ver1";
     publication_info.alloced_by_dbsync = 0;
     ```

     This is equivalent to specifying `-n template_p1,template_p2` on the dbmlsync command line.

The associated script version specified using the ext_opt field, provides the same functionality as the dbmlsync -eu option.

See "-eu option" on page 153.

- 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 option on the dbmlsync command line.

See "-n option" on page 159 and "a_syncpub structure" [*SQL Anywhere Server - Programming*].

6. Run dbmlsync 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 dbmlsync 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

Contents

Introduction to scripted upload

Scripted upload applies only to MobiLink applications that use SQL Anywhere remote databases.

Warning
When you implement scripted upload, dbmlsync 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 rare 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 chapter. 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 383.

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 386.

When using scripted upload, it is strongly recommended that you use the default setting for the dbmlsync extended option LockTables.

You can avoid many problems with scripted uploads by using the default setting for LockTables, which causes dbmlsync 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 388

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 388.

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, in most cases it doesn't 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" [*MobiLink - Server Administration*].

Locking

You can avoid many problems with scripted uploads by using the default setting for the dbmlsync extended option LockTables, which causes dbmlsync 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 379.

Redundant uploads

In most cases, 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 dbmlsync began building the last successful upload. This progress value can be overridden with a different value using the sp_hook_dbmlsync_set_upload_end_progress hook.

See "sp_hook_dbmlsync_set_upload_end_progress" on page 295.

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 can only tell whether an upload was successfully applied to the consolidated database by using 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. Dbmlsync prevents this data loss if each upload built by your upload procedures includes all changes that were committed in the remote database before the sp_hook_dbmlsync_set_upload_end_progress hook was called.

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.

Time	
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.

Dbmlsync uses several mechanisms to ensure that the download does not overwrite any change the was uncommitted when the sp_hook_dbmlsync_set_upload_end_progress hook was called or was committed after the sp_hook_dbmlsync_set_upload_end_progress hook was called.

Any change committed before the hook was called is not protected and may be overwritten when the download is applied. However, as long as the change was included in the upload (which is sent before the download is built) the change is sent to the MobiLink server and your server-side scripts are able to resolve it with the data in the consolidated database before the download is built.

Scripted upload with no table locking

By default, dbmlsync 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 dbmlsync 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.
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_dbmlsync_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_dbmlsync_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.
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.)
publication_*n*	publication name	The publications being synchronized, where *n* is an integer. The numbering of *n* 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 239.

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 295
- “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 388.

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 388.

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 WITH SCRIPTED UPLOAD p1(
   // 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 388.

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.

(In this example, we specify 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;eng=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" %sqlany11%\MobiLink\setup\syncsa.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 ( ?, ?, ? )' );

CALL ml_add_table_script( 'default', 'employee', 'upload_update',
       'UPDATE employee SET name = ?, salary = ? WHERE id = ?' );

CALL ml_add_table_script( 'default', 'employee', 'upload_delete',
       'DELETE FROM employee WHERE 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;eng=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 dbmlsync 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 dbmlsync extended option LockTables, which causes dbmlsync 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 dbmlsync 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';
```

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:

```
mlsrv11 -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 eng=consol
dbstop -y -c eng=remote

dberase -y consol.db
dberase -y remote.db

del remote.mlc mlserver.mls
```

Glossary

Glossary

Adaptive Server Anywhere (ASA)

The relational database server component of SQL Anywhere Studio, intended for use in mobile and embedded environments or as a server for small and medium-sized businesses. In version 10.0.0, Adaptive Server Anywhere was renamed SQL Anywhere Server, and SQL Anywhere Studio was renamed SQL Anywhere.

See also: “SQL Anywhere” on page 421.

agent ID

See also: “client message store ID” on page 399.

article

In MobiLink or SQL Remote, an article is a database object that represents a whole table, or a subset of the columns and rows in a table. Articles are grouped together in a publication.

See also:

- “replication” on page 419
- “publication” on page 416

atomic transaction

A transaction that is guaranteed to complete successfully or not at all. If an error prevents part of an atomic transaction from completing, the transaction is rolled back to prevent the database from being left in an inconsistent state.

base table

Permanent tables for data. Tables are sometimes called **base tables** to distinguish them from temporary tables and views.

See also:

- “temporary table” on page 423
- “view” on page 425

bit array

A bit array is a type of array data structure that is used for efficient storage of a sequence of bits. A bit array is similar to a character string, except that the individual pieces are 0s (zeros) and 1s (ones) instead of characters. Bit arrays are typically used to hold a string of Boolean values.

business rule

A guideline based on real-world requirements. Business rules are typically implemented through check constraints, user-defined data types, and the appropriate use of transactions.

See also:

- "constraint" on page 401
- "user-defined data type" on page 425

carrier

A MobiLink object, stored in MobiLink system tables or a Notifier properties file, that contains information about a public carrier for use by server-initiated synchronization.

See also: "server-initiated synchronization" on page 420.

character set

A character set is a set of symbols, including letters, digits, spaces, and other symbols. An example of a character set is ISO-8859-1, also known as Latin1.

See also:

- "code page" on page 399
- "encoding" on page 405
- "collation" on page 399

check constraint

A restriction that enforces specified conditions on a column or set of columns.

See also:

- "constraint" on page 401
- "foreign key constraint" on page 406
- "primary key constraint" on page 416
- "unique constraint" on page 424

checkpoint

The point at which all changes to the database are saved to the database file. At other times, committed changes are saved only to the transaction log.

checksum

The calculated number of bits of a database page that is recorded with the database page itself. The checksum allows the database management system to validate the integrity of the page by ensuring that the numbers match as the page is being written to disk. If the counts match, it's assumed that page was successfully written.

client message store

In QAnywhere, a SQL Anywhere database on the remote device that stores messages.

client message store ID

In QAnywhere, a MobiLink remote ID that uniquely identifies a client message store.

client/server

A software architecture where one application (the client) obtains information from and sends information to another application (the server). The two applications often reside on different computers connected by a network.

code page

A code page is an encoding that maps characters of a character set to numeric representations, typically an integer between 0 and 255. An example of a code page is Windows code page 1252. For the purposes of this documentation, code page and encoding are interchangeable terms.

See also:

- "character set" on page 398
- "encoding" on page 405
- "collation" on page 399

collation

A combination of a character set and a sort order that defines the properties of text in the database. For SQL Anywhere databases, the default collation is determined by the operating system and language on which the server is running; for example, the default collation on English Windows systems is 1252LATIN1. A collation, also called a collating sequence, is used for comparing and sorting strings.

See also:

- "character set" on page 398
- "code page" on page 399
- "encoding" on page 405

command file

A text file containing SQL statements. Command files can be built manually, or they can be built automatically by database utilities. The dbunload utility, for example, creates a command file consisting of the SQL statements necessary to recreate a given database.

communication stream

In MobiLink, the network protocol used for communication between the MobiLink client and the MobiLink server.

concurrency

The simultaneous execution of two or more independent, and possibly competing, processes. SQL Anywhere automatically uses locking to isolate transactions and ensure that each concurrent application sees a consistent set of data.

See also:

- “transaction” on page 423
- “isolation level” on page 409

conflict resolution

In MobiLink, conflict resolution is logic that specifies what to do when two users modify the same row on different remote databases.

connection ID

A unique number that identifies a given connection between a client application and the database. You can determine the current connection ID using the following SQL statement:

```
SELECT CONNECTION_PROPERTY( 'Number' );
```

connection-initiated synchronization

A form of MobiLink server-initiated synchronization in which synchronization is initiated when there are changes to connectivity.

See also: “server-initiated synchronization” on page 420.

connection profile

A set of parameters that are required to connect to a database, such as user name, password, and server name, that is stored and used as a convenience.

consolidated database

In distributed database environments, a database that stores the master copy of the data. In case of conflict or discrepancy, the consolidated database is considered to have the primary copy of the data.

See also:

- “synchronization” on page 423
- “replication” on page 419

constraint

A restriction on the values contained in a particular database object, such as a table or column. For example, a column may have a uniqueness constraint, which requires that all values in the column be different. A table may have a foreign key constraint, which specifies how the information in the table relates to data in some other table.

See also:

- "check constraint" on page 398
- "foreign key constraint" on page 406
- "primary key constraint" on page 416
- "unique constraint" on page 424

contention

The act of competing for resources. For example, in database terms, two or more users trying to edit the same row of a database contend for the rights to edit that row.

correlation name

The name of a table or view that is used in the FROM clause of a query—either its original name, or an alternate name, that is defined in the FROM clause.

creator ID

In UltraLite Palm OS applications, an ID that is assigned when the application is created.

cursor

A named linkage to a result set, used to access and update rows from a programming interface. In SQL Anywhere, cursors support forward and backward movement through the query results. Cursors consist of two parts: the cursor result set, typically defined by a SELECT statement; and the cursor position.

See also:

- "cursor result set" on page 401
- "cursor position" on page 401

cursor position

A pointer to one row within the cursor result set.

See also:

- "cursor" on page 401
- "cursor result set" on page 401

cursor result set

The set of rows resulting from a query that is associated with a cursor.

See also:

- “cursor” on page 401
- “cursor position” on page 401

data cube

A multi-dimensional result set with each dimension reflecting a different way to group and sort the same results. Data cubes provide complex information about data that would otherwise require self-join queries and correlated subqueries. Data cubes are a part of OLAP functionality.

data definition language (DDL)

The subset of SQL statements for defining the structure of data in the database. DDL statements create, modify, and remove database objects, such as tables and users.

data manipulation language (DML)

The subset of SQL statements for manipulating data in the database. DML statements retrieve, insert, update, and delete data in the database.

data type

The format of data, such as CHAR or NUMERIC. In the ANSI SQL standard, data types can also include a restriction on size, character set, and collation.

See also: “domain” on page 404.

database

A collection of tables that are related by primary and foreign keys. The tables hold the information in the database. The tables and keys together define the structure of the database. A database management system accesses this information.

See also:

- “foreign key” on page 406
- “primary key” on page 416
- “database management system (DBMS)” on page 403
- “relational database management system (RDBMS)” on page 418

database administrator (DBA)

The user with the permissions required to maintain the database. The DBA is generally responsible for all changes to a database schema, and for managing users and groups. The role of database administrator is automatically built into databases as user ID DBA with password sql.

database connection

A communication channel between a client application and the database. A valid user ID and password are required to establish a connection. The privileges granted to the user ID determine the actions that can be carried out during the connection.

database file

A database is held in one or more database files. There is an initial file, and subsequent files are called dbspaces. Each table, including its indexes, must be contained within a single database file.

See also: "dbspace" on page 404.

database management system (DBMS)

A collection of programs that allow you to create and use databases.

See also: "relational database management system (RDBMS)" on page 418.

database name

The name given to a database when it is loaded by a server. The default database name is the root of the initial database file.

See also: "database file" on page 403.

database object

A component of a database that contains or receives information. Tables, indexes, views, procedures, and triggers are database objects.

database owner (dbo)

A special user that owns the system objects not owned by SYS.

See also:

- "database administrator (DBA)" on page 402
- "SYS" on page 423

database server

A computer program that regulates all access to information in a database. SQL Anywhere provides two types of servers: network servers and personal servers.

DBA authority

The level of permission that enables a user to do administrative activity in the database. The DBA user has DBA authority by default.

See also: "database administrator (DBA)" on page 402.

dbspace

An additional database file that creates more space for data. A database can be held in up to 13 separate files (an initial file and 12 dbspaces). Each table, together with its indexes, must be contained in a single database file. The SQL command CREATE DBSPACE adds a new file to the database.

See also: "database file" on page 403.

deadlock

A state where a set of transactions arrives at a place where none can proceed.

device tracking

In MobiLink server-initiated synchronization, functionality that allows you to address messages using the MobiLink user name that identifies a device.

See also: "server-initiated synchronization" on page 420.

direct row handling

In MobiLink, a way to synchronize table data to sources other than the MobiLink-supported consolidated databases. You can implement both uploads and downloads with direct row handling.

See also:

- "consolidated database" on page 400
- "SQL-based synchronization" on page 421

domain

Aliases for built-in data types, including precision and scale values where applicable, and optionally including DEFAULT values and CHECK conditions. Some domains, such as the monetary data types, are pre-defined in SQL Anywhere. Also called user-defined data type.

See also: "data type" on page 402.

download

The stage in synchronization where data is transferred from the consolidated database to a remote database.

dynamic SQL

SQL that is generated programmatically by your program before it is executed. UltraLite dynamic SQL is a variant designed for small-footprint devices.

EBF

Express Bug Fix. An express bug fix is a subset of the software with one or more bug fixes. The bug fixes are listed in the release notes for the update. Bug fix updates may only be applied to installed software with the same version number. Some testing has been performed on the software, but the software has not undergone full testing. You should not distribute these files with your application unless you have verified the suitability of the software yourself.

embedded SQL

A programming interface for C programs. SQL Anywhere embedded SQL is an implementation of the ANSI and IBM standard.

encoding

Also known as character encoding, an encoding is a method by which each character in a character set is mapped onto one or more bytes of information, typically represented as a hexadecimal number. An example of an encoding is UTF-8.

See also:

- "character set" on page 398
- "code page" on page 399
- "collation" on page 399

event model

In MobiLink, the sequence of events that make up a synchronization, such as begin_synchronization and download_cursor. Events are invoked if a script is created for them.

external login

An alternate login name and password used when communicating with a remote server. By default, SQL Anywhere uses the names and passwords of its clients whenever it connects to a remote server on behalf of those clients. However, this default can be overridden by creating external logins. External logins are alternate login names and passwords used when communicating with a remote server.

extraction

In SQL Remote replication, the act of unloading the appropriate structure and data from the consolidated database. This information is used to initialize the remote database.

See also: "replication" on page 419.

failover

Switching to a redundant or standby server, system, or network on failure or unplanned termination of the active server, system, or network. Failover happens automatically.

FILE

In SQL Remote replication, a message system that uses shared files for exchanging replication messages. This is useful for testing and for installations without an explicit message-transport system.

See also:"replication" on page 419.

file-based download

In MobiLink, a way to synchronize data in which downloads are distributed as files, allowing offline distribution of synchronization changes.

file-definition database

In MobiLink, a SQL Anywhere database that is used for creating download files.

See also: "file-based download" on page 406.

foreign key

One or more columns in a table that duplicate the primary key values in another table. Foreign keys establish relationships between tables.

See also:

- "primary key" on page 416
- "foreign table" on page 406

foreign key constraint

A restriction on a column or set of columns that specifies how the data in the table relates to the data in some other table. Imposing a foreign key constraint on a set of columns makes those columns the foreign key.

See also:

- "constraint" on page 401
- "check constraint" on page 398
- "primary key constraint" on page 416
- "unique constraint" on page 424

foreign table

The table containing the foreign key.

See also: "foreign key" on page 406.

full backup

A backup of the entire database, and optionally, the transaction log. A full backup contains all the information in the database and provides protection in the event of a system or media failure.

See also: "incremental backup" on page 408.

gateway

A MobiLink object, stored in MobiLink system tables or a Notifier properties file, that contains information about how to send messages for server-initiated synchronization.

See also: "server-initiated synchronization" on page 420.

generated join condition

A restriction on join results that is automatically generated. There are two types: key and natural. Key joins are generated when you specify KEY JOIN or when you specify the keyword JOIN but do not use the keywords CROSS, NATURAL, or ON. For a key join, the generated join condition is based on foreign key relationships between tables. Natural joins are generated when you specify NATURAL JOIN; the generated join condition is based on common column names in the two tables.

See also:

- "join" on page 410
- "join condition" on page 410

generation number

In MobiLink, a mechanism for forcing remote databases to upload data before applying any more download files.

See also: "file-based download" on page 406.

global temporary table

A type of temporary table for which data definitions are visible to all users until explicitly dropped. Global temporary tables let each user open their own identical instance of a table. By default, rows are deleted on commit, and rows are always deleted when the connection is ended.

See also:

- "temporary table" on page 423
- "local temporary table" on page 410

grant option

The level of permission that allows a user to grant permissions to other users.

hash

A hash is an index optimization that transforms index entries into keys. An index hash aims to avoid the expensive operation of finding, loading, and then unpacking the rows to determine the indexed value, by including enough of the actual row data with its row ID.

histogram

The most important component of column statistics, histograms are a representation of data distribution. SQL Anywhere maintains histograms to provide the optimizer with statistical information about the distribution of values in columns.

iAnywhere JDBC driver

The iAnywhere JDBC driver provides a JDBC driver that has some performance benefits and feature benefits compared to the pure Java jConnect JDBC driver, but which is not a pure-Java solution. The iAnywhere JDBC driver is recommended in most cases.

See also:

- "JDBC" on page 409
- "jConnect" on page 409

identifier

A string of characters used to reference a database object, such as a table or column. An identifier may contain any character from A through Z, a through z, 0 through 9, underscore (_), at sign (@), number sign (#), or dollar sign ($).

incremental backup

A backup of the transaction log only, typically used between full backups.

See also: "transaction log" on page 423.

index

A sorted set of keys and pointers associated with one or more columns in a base table. An index on one or more columns of a table can improve performance.

InfoMaker

A reporting and data maintenance tool that lets you create sophisticated forms, reports, graphs, cross-tabs, and tables, and applications that use these reports as building blocks.

inner join

A join in which rows appear in the result set only if both tables satisfy the join condition. Inner joins are the default.

See also:

- "join" on page 410
- "outer join" on page 414

integrated login

A login feature that allows the same single user ID and password to be used for operating system logins, network logins, and database connections.

integrity

Adherence to rules that ensure that data is correct and accurate, and that the relational structure of the database is intact.

See also: "referential integrity" on page 418.

Interactive SQL

A SQL Anywhere application that allows you to query and alter data in your database, and modify the structure of your database. Interactive SQL provides a pane for you to enter SQL statements, and panes that return information about how the query was processed and the result set.

isolation level

The degree to which operations in one transaction are visible to operations in other concurrent transactions. There are four isolation levels, numbered 0 through 3. Level 3 provides the highest level of isolation. Level 0 is the default setting. SQL Anywhere also supports three snapshot isolation levels: snapshot, statement-snapshot, and readonly-statement-snapshot.

See also: "snapshot isolation" on page 421.

JAR file

Java archive file. A compressed file format consisting of a collection of one or more packages used for Java applications. It includes all the resources necessary to install and run a Java program in a single compressed file.

Java class

The main structural unit of code in Java. It is a collection of procedures and variables grouped together because they all relate to a specific, identifiable category.

jConnect

A Java implementation of the JavaSoft JDBC standard. It provides Java developers with native database access in multi-tier and heterogeneous environments. However, the iAnywhere JDBC driver is the preferred JDBC driver for most cases.

See also:

- "JDBC" on page 409
- "iAnywhere JDBC driver" on page 408

JDBC

Java Database Connectivity. A SQL-language programming interface that allows Java applications to access relational data. The preferred JDBC driver is the iAnywhere JDBC driver.

See also:

- “jConnect” on page 409
- “iAnywhere JDBC driver” on page 408

join

A basic operation in a relational system that links the rows in two or more tables by comparing the values in specified columns.

join condition

A restriction that affects join results. You specify a join condition by inserting an ON clause or WHERE clause immediately after the join. In the case of natural and key joins, SQL Anywhere generates a join condition.

See also:

- “join” on page 410
- “generated join condition” on page 407

join type

SQL Anywhere provides four types of joins: cross join, key join, natural join, and joins using an ON clause.

See also: “join” on page 410.

light weight poller

In MobiLink server-initiated synchronization, a device application that polls for push notifications from a MobiLink server.

See also: “server-initiated synchronization” on page 420.

Listener

A program, dblsn, that is used for MobiLink server-initiated synchronization. Listeners are installed on remote devices and configured to initiate actions on the device when they receive push notifications.

See also: “server-initiated synchronization” on page 420.

local temporary table

A type of temporary table that exists only for the duration of a compound statement or until the end of the connection. Local temporary tables are useful when you need to load a set of data only once. By default, rows are deleted on commit.

See also:

- “temporary table” on page 423
- “global temporary table” on page 407

lock

A concurrency control mechanism that protects the integrity of data during the simultaneous execution of multiple transactions. SQL Anywhere automatically applies locks to prevent two connections from changing the same data at the same time, and to prevent other connections from reading data that is in the process of being changed.

You control locking by setting the isolation level.

See also:

- "isolation level" on page 409
- "concurrency" on page 400
- "integrity" on page 409

log file

A log of transactions maintained by SQL Anywhere. The log file is used to ensure that the database is recoverable in the event of a system or media failure, to improve database performance, and to allow data replication using SQL Remote.

See also:

- "transaction log" on page 423
- "transaction log mirror" on page 424
- "full backup" on page 406

logical index

A reference (pointer) to a physical index. There is no indexing structure stored on disk for a logical index.

LTM

Log Transfer Manager (LTM) also called Replication Agent. Used with Replication Server, the LTM is the program that reads a database transaction log and sends committed changes to Sybase Replication Server.

See: "Replication Server" on page 419.

maintenance release

A maintenance release is a complete set of software that upgrades installed software from an older version with the same major version number (version number format is *major.minor.patch.build*). Bug fixes and other changes are listed in the release notes for the upgrade.

materialized view

A materialized view is a view that has been computed and stored on disk. Materialized views have characteristics of both views (they are defined using a query specification), and of tables (they allow most table operations to be performed on them).

See also:

- “base table” on page 397
- “view” on page 425

message log

A log where messages from an application such as a database server or MobiLink server can be stored. This information can also appear in a messages window or be logged to a file. The message log includes informational messages, errors, warnings, and messages from the MESSAGE statement.

message store

In QAnywhere, databases on the client and server device that store messages.

See also:

- “client message store” on page 399
- “server message store” on page 421

message system

In SQL Remote replication, a protocol for exchanging messages between the consolidated database and a remote database. SQL Anywhere includes support for the following message systems: FILE, FTP, and SMTP.

See also:

- “replication” on page 419
- “FILE” on page 405

message type

In SQL Remote replication, a database object that specifies how remote users communicate with the publisher of a consolidated database. A consolidated database may have several message types defined for it; this allows different remote users to communicate with it using different message systems.

See also:

- “replication” on page 419
- “consolidated database” on page 400

metadata

Data about data. Metadata describes the nature and content of other data.

See also: “schema” on page 420.

mirror log

See also: “transaction log mirror” on page 424.

MobiLink

A session-based synchronization technology designed to synchronize UltraLite and SQL Anywhere remote databases with a consolidated database.

See also:

- “consolidated database” on page 400
- “synchronization” on page 423
- “UltraLite” on page 424

MobiLink client

There are two kinds of MobiLink clients. For SQL Anywhere remote databases, the MobiLink client is the dbmlsync command line utility. For UltraLite remote databases, the MobiLink client is built in to the UltraLite runtime library.

MobiLink Monitor

A graphical tool for monitoring MobiLink synchronizations.

MobiLink server

The computer program that runs MobiLink synchronization, mlsrv11.

MobiLink system table

System tables that are required by MobiLink synchronization. They are installed by MobiLink setup scripts into the MobiLink consolidated database.

MobiLink user

A MobiLink user is used to connect to the MobiLink server. You create the MobiLink user on the remote database and register it in the consolidated database. MobiLink user names are entirely independent of database user names.

network protocol

The type of communication, such as TCP/IP or HTTP.

network server

A database server that accepts connections from computers sharing a common network.

See also: “personal server” on page 415.

normalization

The refinement of a database schema to eliminate redundancy and improve organization according to rules based on relational database theory.

Notifier

A program that is used by MobiLink server-initiated synchronization. Notifiers are integrated into the MobiLink server. They check the consolidated database for push requests, and send push notifications.

See also:

- "server-initiated synchronization" on page 420
- "Listener" on page 410

object tree

In Sybase Central, the hierarchy of database objects. The top level of the object tree shows all products that your version of Sybase Central supports. Each product expands to reveal its own sub-tree of objects.

See also: "Sybase Central" on page 422.

ODBC

Open Database Connectivity. A standard Windows interface to database management systems. ODBC is one of several interfaces supported by SQL Anywhere.

ODBC Administrator

A Microsoft program included with Windows operating systems for setting up ODBC data sources.

ODBC data source

A specification of the data a user wants to access via ODBC, and the information needed to get to that data.

outer join

A join that preserves all the rows in a table. SQL Anywhere supports left, right, and full outer joins. A left outer join preserves the rows in the table to the left of the join operator, and returns a null when a row in the right table does not satisfy the join condition. A full outer join preserves all the rows from both tables.

See also:

- "join" on page 410
- "inner join" on page 408

package

In Java, a collection of related classes.

parse tree

An algebraic representation of a query.

PDB

A Palm database file.

performance statistic

A value reflecting the performance of the database system. The CURRREAD statistic, for example, represents the number of file reads issued by the database server that have not yet completed.

personal server

A database server that runs on the same computer as the client application. A personal database server is typically used by a single user on a single computer, but it can support several concurrent connections from that user.

physical index

The actual indexing structure of an index, as it is stored on disk.

plug-in module

In Sybase Central, a way to access and administer a product. Plug-ins are usually installed and registered automatically with Sybase Central when you install the respective product. Typically, a plug-in appears as a top-level container, in the Sybase Central main window, using the name of the product itself; for example, SQL Anywhere.

See also: "Sybase Central" on page 422.

policy

In QAnywhere, the way you specify when message transmission should occur.

polling

In MobiLink server-initiated synchronization, the way a light weight poller, such as the MobiLink Listener, requests push notifications from a Notifier.

See also: "server-initiated synchronization" on page 420.

PowerDesigner

A database modeling application. PowerDesigner provides a structured approach to designing a database or data warehouse. SQL Anywhere includes the Physical Data Model component of PowerDesigner.

PowerJ

A Sybase product for developing Java applications.

predicate

A conditional expression that is optionally combined with the logical operators AND and OR to make up the set of conditions in a WHERE or HAVING clause. In SQL, a predicate that evaluates to UNKNOWN is interpreted as FALSE.

primary key

A column or list of columns whose values uniquely identify every row in the table.

See also: “foreign key” on page 406.

primary key constraint

A uniqueness constraint on the primary key columns. A table can have only one primary key constraint.

See also:

- “constraint” on page 401
- “check constraint” on page 398
- “foreign key constraint” on page 406
- “unique constraint” on page 424
- “integrity” on page 409

primary table

The table containing the primary key in a foreign key relationship.

proxy table

A local table containing metadata used to access a table on a remote database server as if it were a local table.

See also: “metadata” on page 412.

publication

In MobiLink or SQL Remote, a database object that identifies data that is to be synchronized. In MobiLink, publications exist only on the clients. A publication consists of articles. SQL Remote users can receive a publication by subscribing to it. MobiLink users can synchronize a publication by creating a synchronization subscription to it.

See also:

- “replication” on page 419
- “article” on page 397
- “publication update” on page 416

publication update

In SQL Remote replication, a list of changes made to one or more publications in one database. A publication update is sent periodically as part of a replication message to the remote database(s).

See also:

- “replication” on page 419
- “publication” on page 416

publisher

In SQL Remote replication, the single user in a database who can exchange replication messages with other replicating databases.

See also: “replication” on page 419.

push notification

In QAnywhere, a special message delivered from the server to a QAnywhere client that prompts the client to initiate a message transmission. In MobiLink server-initiated synchronization, a special message delivered from a Notifer to a device that contains push request data and internal information.

See also:

- “QAnywhere” on page 417
- “server-initiated synchronization” on page 420

push request

In MobiLink server-initiated synchronization, a row of values in a result set that a Notifier checks to determine if push notifications need to be sent to a device.

See also: “server-initiated synchronization” on page 420.

QAnywhere

Application-to-application messaging, including mobile device to mobile device and mobile device to and from the enterprise, that permits communication between custom programs running on mobile or wireless devices and a centrally located server application.

QAnywhere agent

In QAnywhere, a process running on the client device that monitors the client message store and determines when message transmission should occur.

query

A SQL statement or group of SQL statements that access and/or manipulate data in a database.

See also: “SQL” on page 421.

Redirector

A web server plug-in that routes requests and responses between a client and the MobiLink server. This plug-in also implements load-balancing and failover mechanisms.

reference database

In MobiLink, a SQL Anywhere database used in the development of UltraLite clients. You can use a single SQL Anywhere database as both reference and consolidated database during development. Databases made with other products cannot be used as reference databases.

referencing object

An object, such as a view, whose definition directly references another object in the database, such as a table.

See also: “foreign key” on page 406.

referenced object

An object, such as a table, that is directly referenced in the definition of another object, such as a view.

See also: “primary key” on page 416.

referential integrity

Adherence to rules governing data consistency, specifically the relationships between the primary and foreign key values in different tables. To have referential integrity, the values in each foreign key must correspond to the primary key values of a row in the referenced table.

See also:

- “primary key” on page 416
- “foreign key” on page 406

regular expression

A regular expression is a sequence of characters, wildcards, and operators that defines a pattern to search for within a string.

relational database management system (RDBMS)

A type of database management system that stores data in the form of related tables.

See also: “database management system (DBMS)” on page 403.

remote database

In MobiLink or SQL Remote, a database that exchanges data with a consolidated database. Remote databases may share all or some of the data in the consolidated database.

See also:

- “synchronization” on page 423
- “consolidated database” on page 400

REMOTE DBA authority

In SQL Remote, a level of permission required by the Message Agent (dbremote). In MobiLink, a level of permission required by the SQL Anywhere synchronization client (dbmlsync). When the Message Agent (dbremote) or synchronization client connects as a user who has this authority, it has full DBA access. The user ID has no additional permissions when not connected through the Message Agent (dbremote) or synchronization client (dbmlsync).

See also: “DBA authority” on page 403.

remote ID

A unique identifier in SQL Anywhere and UltraLite databases that is used by MobiLink. The remote ID is initially set to NULL and is set to a GUID during a database's first synchronization.

replication

The sharing of data among physically distinct databases. Sybase has three replication technologies: MobiLink, SQL Remote, and Replication Server.

Replication Agent

See: “LTM” on page 411.

replication frequency

In SQL Remote replication, a setting for each remote user that determines how often the publisher's message agent should send replication messages to that remote user.

See also: “replication” on page 419.

replication message

In SQL Remote or Replication Server, a communication sent between a publishing database and a subscribing database. Messages contain data, passthrough statements, and information required by the replication system.

See also:

- “replication” on page 419
- “publication update” on page 416

Replication Server

A Sybase connection-based replication technology that works with SQL Anywhere and Adaptive Server Enterprise. It is intended for near-real time replication between a few databases.

See also: “LTM” on page 411.

role

In conceptual database modeling, a verb or phrase that describes a relationship from one point of view. You can describe each relationship with two roles. Examples of roles are "contains" and "is a member of."

role name

The name of a foreign key. This is called a role name because it names the relationship between the foreign table and primary table. By default, the role name is the table name, unless another foreign key is already using that name, in which case the default role name is the table name followed by a three-digit unique number. You can also create the role name yourself.

See also: “foreign key” on page 406.

rollback log

A record of the changes made during each uncommitted transaction. In the event of a ROLLBACK request or a system failure, uncommitted transactions are reversed out of the database, returning the database to its former state. Each transaction has a separate rollback log, which is deleted when the transaction is complete.

See also: “transaction” on page 423.

row-level trigger

A trigger that executes once for each row that is changed.

See also:

- “trigger” on page 424
- “statement-level trigger” on page 422

schema

The structure of a database, including tables, columns, and indexes, and the relationships between them.

script

In MobiLink, code written to handle MobiLink events. Scripts programmatically control data exchange to meet business needs.

See also: “event model” on page 405.

script-based upload

In MobiLink, a way to customize the upload process as an alternative to using the log file.

script version

In MobiLink, a set of synchronization scripts that are applied together to create a synchronization.

secured feature

A feature specified by the -sf option when a database server is started, so it is not available for any database running on that database server.

server-initiated synchronization

A way to initiate MobiLink synchronization from the MobiLink server.

server management request

A QAnywhere message that is formatted as XML and sent to the QAnywhere system queue as a way to administer the server message store or monitor QAnywhere applications.

server message store

In QAnywhere, a relational database on the server that temporarily stores messages until they are transmitted to a client message store or JMS system. Messages are exchanged between clients via the server message store.

service

In Windows operating systems, a way of running applications when the user ID running the application is not logged on.

session-based synchronization

A type of synchronization where synchronization results in consistent data representation across both the consolidated and remote databases. MobiLink is session-based.

snapshot isolation

A type of isolation level that returns a committed version of the data for transactions that issue read requests. SQL Anywhere provides three snapshot isolation levels: snapshot, statement-snapshot, and readonly-statement-snapshot. When using snapshot isolation, read operations do not block write operations.

See also: “isolation level” on page 409.

SQL

The language used to communicate with relational databases. ANSI has defined standards for SQL, the latest of which is SQL-2003. SQL stands, unofficially, for Structured Query Language.

SQL Anywhere

The relational database server component of SQL Anywhere that is intended for use in mobile and embedded environments or as a server for small and medium-sized businesses. SQL Anywhere is also the name of the package that contains the SQL Anywhere RDBMS, the UltraLite RDBMS, MobiLink synchronization software, and other components.

SQL-based synchronization

In MobiLink, a way to synchronize table data to MobiLink-supported consolidated databases using MobiLink events. For SQL-based synchronization, you can use SQL directly or you can return SQL using the MobiLink server APIs for Java and .NET.

SQL Remote

A message-based data replication technology for two-way replication between consolidated and remote databases. The consolidated and remote databases must be SQL Anywhere.

SQL statement

A string containing SQL keywords designed for passing instructions to a DBMS.

See also:

- “schema” on page 420
- “SQL” on page 421
- “database management system (DBMS)” on page 403

statement-level trigger

A trigger that executes after the entire triggering statement is completed.

See also:

- “trigger” on page 424
- “row-level trigger” on page 420

stored procedure

A stored procedure is a group of SQL instructions stored in the database and used to execute a set of operations or queries on a database server

string literal

A string literal is a sequence of characters enclosed in single quotes.

subquery

A SELECT statement that is nested inside another SELECT, INSERT, UPDATE, or DELETE statement, or another subquery.

There are two types of subquery: correlated and nested.

subscription

In MobiLink synchronization, a link in a client database between a publication and a MobiLink user, allowing the data described by the publication to be synchronized.

In SQL Remote replication, a link between a publication and a remote user, allowing the user to exchange updates on that publication with the consolidated database.

See also:

- “publication” on page 416
- “MobiLink user” on page 413

Sybase Central

A database management tool that provides SQL Anywhere database settings, properties, and utilities in a graphical user interface. Sybase Central can also be used for managing other Sybase products, including MobiLink.

synchronization

The process of replicating data between databases using MobiLink technology.

In SQL Remote, synchronization is used exclusively to denote the process of initializing a remote database with an initial set of data.

See also:

- "MobiLink" on page 413
- "SQL Remote" on page 421

SYS

A special user that owns most of the system objects. You cannot log in as SYS.

system object

Database objects owned by SYS or dbo.

system table

A table, owned by SYS or dbo, that holds metadata. System tables, also known as data dictionary tables, are created and maintained by the database server.

system view

A type of view, included in every database, that presents the information held in the system tables in an easily understood format.

temporary table

A table that is created for the temporary storage of data. There are two types: global and local.

See also:

- "local temporary table" on page 410
- "global temporary table" on page 407

transaction

A sequence of SQL statements that comprise a logical unit of work. A transaction is processed in its entirety or not at all. SQL Anywhere supports transaction processing, with locking features built in to allow concurrent transactions to access the database without corrupting the data. Transactions end either with a COMMIT statement, which makes the changes to the data permanent, or a ROLLBACK statement, which undoes all the changes made during the transaction.

transaction log

A file storing all changes made to a database, in the order in which they are made. It improves performance and allows data recovery in the event the database file is damaged.

transaction log mirror

An optional identical copy of the transaction log file, maintained simultaneously. Every time a database change is written to the transaction log file, it is also written to the transaction log mirror file.

A mirror file should be kept on a separate device from the transaction log, so that if either device fails, the other copy of the log keeps the data safe for recovery.

See also: “transaction log” on page 423.

transactional integrity

In MobiLink, the guaranteed maintenance of transactions across the synchronization system. Either a complete transaction is synchronized, or no part of the transaction is synchronized.

transmission rule

In QAnywhere, logic that determines when message transmission is to occur, which messages to transmit, and when messages should be deleted.

trigger

A special form of stored procedure that is executed automatically when a user runs a query that modifies the data.

See also:

- “row-level trigger” on page 420
- “statement-level trigger” on page 422
- “integrity” on page 409

UltraLite

A database optimized for small, mobile, and embedded devices. Intended platforms include cell phones, pagers, and personal organizers.

UltraLite runtime

An in-process relational database management system that includes a built-in MobiLink synchronization client. The UltraLite runtime is included in the libraries used by each of the UltraLite programming interfaces, and in the UltraLite engine.

unique constraint

A restriction on a column or set of columns requiring that all non-null values are different. A table can have multiple unique constraints.

See also:

- “foreign key constraint” on page 406
- “primary key constraint” on page 416
- “constraint” on page 401

unload

Unloading a database exports the structure and/or data of the database to text files (SQL command files for the structure, and ASCII comma-separated files for the data). You unload a database with the Unload utility.

In addition, you can unload selected portions of your data using the UNLOAD statement.

upload

The stage in synchronization where data is transferred from a remote database to a consolidated database.

user-defined data type

See "domain" on page 404.

validate

To test for particular types of file corruption of a database, table, or index.

view

A SELECT statement that is stored in the database as an object. It allows users to see a subset of rows or columns from one or more tables. Each time a user uses a view of a particular table, or combination of tables, it is recomputed from the information stored in those tables. Views are useful for security purposes, and to tailor the appearance of database information to make data access straightforward.

window

The group of rows over which an analytic function is performed. A window may contain one, many, or all rows of data that has been partitioned according to the grouping specifications provided in the window definition. The window moves to include the number or range of rows needed to perform the calculations for the current row in the input. The main benefit of the window construct is that it allows additional opportunities for grouping and analysis of results, without having to perform additional queries.

Windows

The Microsoft Windows family of operating systems, such as Windows Vista, Windows XP, and Windows 200x.

Windows CE

See "Windows Mobile" on page 425.

Windows Mobile

A family of operating systems produced by Microsoft for mobile devices.

work table

An internal storage area for interim results during query optimization.

Index