PUBLIC

SQL Anywhere - MobiLink

Document Version: 17.01.0 – 2021-10-15

MobiLink Client Administration

THE BEST RUN SAP

Content

1 MobiLink - Client Administration

The MobiLink data synchronization technology included in the SQL Anywhere product includes a synchronization server, client software for SQL Anywhere and UltraLite databases, and a communication protocol between clients and servers.

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.

In this section:

1.1 MobiLink Clients

The MobiLink server currently supports the use of SQL Anywhere and UltraLite clients.

In this section:

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

Network Protocols for Clients [page 7]
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.

System Tables in MobiLink [page 8]
When you set up a database for use as a consolidated database with MobiLink, the MobiLink system tables that are required by the MobiLink server are created.

1.1.1 SQL Anywhere as a MobiLink Client

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.

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

Related Information

Initiation of Synchronization [page 102]
SQL Anywhere Clients [page 78]
dbmlsync Synchronization Customization [page 112]
Publications [page 85]
MobiLink Users [page 95]
Synchronization Subscription Creation [page 100]
Using a SQL Anywhere Database as a Remote Database [page 79]
MobiLink SQL Anywhere Client Utility (dbmlsync) Syntax [page 115]

1.1.2 UltraLite as a MobiLink Client

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

Related Information

UltraLite as a MobiLink Client

1.1.3 Network Protocols 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 mlsrv17 command line option is:

```
mlsrv17 -c "connection-string" -x protocol( options )
```

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

```
mlsrv17 -c "DSN=SQL Anywhere 17 Demo" -x tcpip
```

You can configure your protocol using options of the form:

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

For example:

```
mlsrv17 -c "DSN=SQL Anywhere 17 Demo" -x tcpip(
   host=localhost;port=2439)
```

Related Information

-x mlsrv17 Option
MobiLink Client Network Protocol Options [page 24]
CommunicationAddress (adr) Extended Option [page 166]
CommunicationType (ctp) Extended Option [page 167]
Stream Parameters Synchronization Parameter
Stream Type Synchronization Parameter
UltraLite Synchronization Utility (ulsync)

1.1.4 System Tables in MobiLink

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

UltraLite system tables

The schema of an UltraLite database is stored in a proprietary format.

SQL Anywhere system tables

SQL Anywhere system tables cannot be accessed directly, but are accessed via system views.

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

- SYSSYNC system view
- SYSPUBLICATION system view
- SYSSUBSCRIPTION system view
- SYSSYNCSCRIPT system view
- SYSSYNCPROFILE system view
- SYSARTICLE system view
- SYSARTICLECOL system view

SQL Anywhere also provides consolidated views that query system views to provide information that you might need.

Related Information

MobiLink Server System Tables
UltraLite System Tables
System Views
Consolidated Views
SYSSYNC System View
SYSPUBLICATION System View
SYSSUBSCRIPTION System View
SYSSYNCSCRIPT System View
SYSSYNCPROFILE System View
SYSARTICLE System View
SYSARTICLECOL System View

1.2 MobiLink Users in a Synchronization System

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

For a user to be part of a synchronization system:

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

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

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

- In a MobiLink synchronization system, no two user names can be the same except for case. For example, you can have user name **aA** or user name **Aa**, but not both.
- For any given user name, you must always use the same case once you start using it. For example, if you add a user as **Aa** and synchronize with it, you must continue to synchronize using **Aa**. Using **aA** will fail.

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

You can also use user names to control the behavior of the MobiLink server. You do so using the *username* parameter in synchronization scripts.

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.

In this section:

Synchronizations from New Users [page 13]
: 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.

End User Passwords [page 14]
: 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.

Password Changes [page 14]
: MobiLink provides a mechanism for end users to change their password. The interface differs between UltraLite and SQL Anywhere clients.

Remote IDs [page 15]
: The **remote ID** uniquely identifies a remote database in a MobiLink synchronization system.

User Authentication Mechanisms [page 18]
: User authentication is one part of a security system for protecting your data.

User Authentication Architecture [page 18]
: 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.

Authentication Process [page 19]
: The following list provides an explanation of the order of events that occur during authentication.

Custom User Authentication [page 20]
: You can choose to use a user authentication mechanism other than the built-in MobiLink mechanism.

Related Information

Remote IDs and MobiLink User Names in Scripts [page 17]
UltraLite Users

1.2.1 MobiLink User Creation and Registration

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 SQL Central or the CREATE SYNCHRONIZATION USER statement.
- For UltraLite remote databases, you set the User Name and Password synchronization parameters.

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.
- Use SQL 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.
- Specify the -zu+ command line option with mlsrv17. In this case, any existing MobiLink users that have not been added to the consolidated database are added when they first synchronize. This option is useful during development but is not recommended for deployed applications.

Related Information

MobiLink Users [page 95]
User Name Synchronization Parameter
Password Synchronization Parameter
MobiLink User Authentication Utility (mluser)
authenticate_user Connection Event
authenticate_user_hashed Connection Event
-zu mlsrv17 Option

1.2.2 Providing an Initial MobiLink Password for a User (SQL Central)

The password for each user is stored with the user name in the ml_user table. SQL Central is a convenient way of adding individual users and passwords.

Context

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

Procedure

1. Double-click your MobiLink project.
2. Double-click *Consolidated Databases* and click the name of your consolidated database.

3. Click *Users*.
4. Click *File* > *New* > *User*.
5. Follow the instructions in the *Create User Wizard*.

Results

The user is created with the specified password.

Related Information

MobiLink User Authentication Utility (mluser)

1.2.3 Providing Initial MobiLink Passwords (mluser Utility)

The password for each user is stored with the user name in the ml_user table. The mluser utility is useful for batch additions.

Context

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

Procedure

1. Create a file with a single user name and password on each line, separated by white space.
2. At a command prompt, run the mluser utility using the -f option if you have a file containing user names and passwords, or the -u and -p options if you are specifying a single user name and password.

Results

The specified users and passwords are created.

Example

In the following 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.

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

Related Information

Providing an Initial MobiLink Password for a User (SQL Central) [page 11]
MobiLink User Authentication Utility (mluser)

1.2.4 Synchronizations from New Users

Ordinarily, each MobiLink client must provide a valid MobiLink user name and password to connect to a MobiLink server. Setting the -zu+ option when you start the MobiLink server allows the server to accept and respond to synchronization requests from unregistered users.

When a request is received from a user not listed in the ml_user table, the request is serviced and the user is added to the ml_user table.

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

New user, no password

If the user supplied no password, then the user name is added to the ml_user table with a null password. This user is allowed to synchronize without a password.

New user, password

If the user supplies a password, then the user name and password are both added to the ml_user table and the new user name becomes a recognized name in your MobiLink system. In future, this user must specify the same password to synchronize.

New user, new password

A new user may provide information in the new password field, or in the password field. In either case, the new password setting overrides the old password setting, and the new user is added to the MobiLink system using the new password. In future, this user must specify the same password to synchronize.

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.

Related Information

-zu mlsrv17 Option

1.2.5 End User 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.

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.

SQL Anywhere

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

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

Related Information

UltraLite Synchronization Parameters
UltraLite Network Protocol Options
-c dbmlsync Option [page 127]
-mp dbmlsync Option [page 139]
MobiLinkPwd (mp) Extended Option [page 181]

1.2.6 Password Changes

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

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 connect window if you do not supply command line parameters.

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.

An initial password can be set in the consolidated database server or on the first synchronization attempt.

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

Related Information

Synchronizations from New Users [page 13]
Providing an Initial MobiLink Password for a User (SQL Central) [page 11]
-mp dbmlsync Option [page 139]
-mn dbmlsync Option [page 138]
Password Synchronization Parameter
New Password Synchronization Parameter

1.2.7 Remote IDs

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

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

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

Remote IDs must be unique

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

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

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

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

> ⚠ Caution
>
> Be careful about reusing remote IDs. If you re-use a remote ID, for example when replacing or restoring a remote database, call the ml_reset_sync_state stored procedure in the consolidated database for that remote ID prior to the first synchronization of the replacement remote database.

In this section:

MobiLink Remote ID Name [page 16]
: 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.

Remote IDs and MobiLink User Names in Scripts [page 17]
: The MobiLink user identifies a person and is used for authentication. The remote ID uniquely identifies a MobiLink remote database.

Related Information

Authentication Parameters
ml_reset_sync_state System Procedure

1.2.7.1 MobiLink Remote ID Name

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.

If you set the remote ID manually and you subsequently recreate the remote database, you must either give the recreated remote database a different name from the old one or use the ml_reset_sync_state stored procedure to reset the state information in the consolidated database for the remote database.

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

Related Information

Remote ID Settings [page 82]
ml_remote_id Option [MobiLink]
UltraLite ml_remote_id Option
ml_reset_sync_state System Procedure

1.2.7.2 Remote IDs and MobiLink User Names in Scripts

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

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

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

If a MobiLink user wants to synchronize data in different databases but each remote database 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.

Related Information

Script Versions

1.2.8 User Authentication Mechanisms

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.

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 or LDAP.

Related Information

Custom User Authentication [page 20]
MobiLink Client/Server Communications Encryption
Database Security
Data Security

1.2.9 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 SQL Central, using the mluser utility, or by specifying -zu+ when you start the MobiLink server.

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.

Related Information

MobiLink User Creation and Registration [page 10]
Custom User Authentication [page 20]

1.2.10 Authentication Process

The following list provides 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:
    1. 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.
    2. 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:
    1. 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.
    2. 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.
    1. 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.
    2. 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.
    3. If the custom scripts were called and the MobiLink user name you provided is in the ml_user table, nothing happens.
    4. 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:
    1. The parameters are passed to the authenticate_parameters script.
    2. 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.

1.2.11 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:

- mlsrv17 -zu+ option
- authenticate_user script or authenticate_user_hashed script
- authenticate_parameters script

The mlsrv17 -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.

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

In this section:

Related Information

-zu mlsrv17 Option
authenticate_user Connection Event
authenticate_user_hashed Connection Event
authenticate_parameters Connection Event

1.2.11.1 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 *%SQLANYSAMP17%*`\MobiLink\JavaAuthentication`. The sample code in *%SQLANYSAMP17%*`\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.

Related Information

.NET Synchronization Example

1.3 MobiLink Client Utilities

The MobiLink File Transfer utility (mlfiletransfer) is available for MobiLink clients.

In this section:

MobiLink File Transfer Utility (mlfiletransfer) [page 22]
Uploads or downloads a file through MobiLink.

Related Information

UltraLite Utilities
MobiLink Utilities
Database Administration Utilities

1.3.1 MobiLink File Transfer Utility (mlfiletransfer)

Uploads or downloads a file through MobiLink.

Syntax

```
mlfiletransfer [ options ] file
```

Option	Description
-ap `param1`, ...	MobiLink authentication parameters.
-g	Shows transfer progress.
-i	Ignore partial transfer from a previous attempt.
-k	Remote key to identify the remote. This is optional.
-lf `filename`	The local name of the file to be transferred. By default, the name as recognized by the server (that is, `file`) is used.
-lp `path`	The local path for the file to be transferred. By default, the local path is the current directory.
-p `password`	The password for the MobiLink user name.
-q	Quiet mode. Messages are not displayed.

Option	Description
-*s*	Upload a file to MobiLink. Download is the default.
-*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.
`file`	The file to be transferred as named on the server. When downloading, MobiLink looks for the file in the `username` subdirectory of the -ftr directory; if it does not find it there, it looks in the -ftr directory. If the file is not in either place, MobiLink generates an error. File names should not include a path and cannot use ellipsis (three dots), comma, forward slash (/) or backslash (\). When uploading, MobiLink looks for the files in the directory specified with the -ftru mlsrv17 option.

Remarks

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

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

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

You can use mlagent as an alternative to mlfiletransfer.

UltraLite users can also use the MLFileDownload and MLFileUpload methods in the UltraLite runtime.

Example

The following command connects the MobiLink server to the SQL Anywhere 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.

```
mlsrv17 -c "DSN=SQL Anywhere 17 CustDB;uid=ml_server;pwd=sql" -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 17.0" -x tcpip notepad.exe
```

Related Information

Authentication Parameters
MobiLink File Transfers
MobiLink Client Network Protocol Options [page 24]
-ftr mlsrv17 Option
-ftru mlsrv17 Option
authenticate_file_transfer Connection Event
mlagent Command

1.4 MobiLink Client Network Protocol Options

MobiLink client utilities use the following MobiLink client network protocol options when connecting to the MobiLink server.

To use client network protocol options with...	See...
dbmlsync	CommunicationAddress (adr) extended option
UltraLite	Stream Parameters synchronization parameter or -x option in UltraLite Synchronization utility (ulsync)
UltraLiteJ	• Network protocol options for UltraLiteJ synchronization streams • StreamHTTPParms interface [UltraLiteJ] • StreamHTTPSParms interface [UltraLiteJ]
Relay Server	Relay Server configuration file
MobiLink Profiler	Starting the MobiLink Monitor
MobiLink file transfer	MobiLink File Transfer utility (mlfiletransfer)
MobiLink Listener	-x in MobiLink Listener utility for Windows devices (dblsn)

In this section:

The network protocol you choose for the synchronization server must match the synchronization protocol used by the client. Connection options for the MobiLink server are set using the -x mlsrv17 option.

allow_unencrypted_basic_proxy_auth MobiLink Client Network Protocol Option [page 31]
: Permit Basic HTTP proxy authentication.

buffer_size MobiLink Client Network Protocol Option [page 31]
: Specify the maximum number of bytes to buffer before writing to the network. For HTTP and HTTPS, this value translates to the maximum HTTP request body size.

certificate_company MobiLink Client Network Protocol Option [page 33]
: If specified, the application only accepts server certificates when the Organization field on the certificate matches this value.

certificate_name MobiLink Client Network Protocol Option [page 34]
: If specified, the application only accepts server certificates when the Common Name field on the certificate matches this value.

certificate_unit MobiLink Client Network Protocol Option [page 36]
: If specified, the application only accepts server certificates when the Organization Unit field on the certificate matches this value.

client_port MobiLink Client Network Protocol Option [page 38]
: Specify a range of client ports for communication.

compression MobiLink Client Network Protocol Option [page 39]
: Sets the type of data compression to be used.

custom_header MobiLink Client Network Protocol Option [page 40]
: Specify a custom HTTP header.

e2ee_public_key MobiLink Client Network Protocol Option [page 41]
: Specify the file containing the server's public key or X.509 certificate for end-to-end encryption.

fips MobiLink Client Network Protocol Option [page 42]
: Use FIPS-certified encryption implementations for TLS encryption and end-to-end encryption.

host MobiLink Client Network Protocol Option [page 44]
: 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.

http_buffer_responses MobiLink Client Network Protocol Option [page 45]
: When on, this option streams HTTP packets from MobiLink into a buffer before processing them, instead of processing the bytes immediately as they are received.

http_password MobiLink Client Network Protocol Option [page 46]
: Authenticate to third-party HTTP servers and gateways using RFC 2617 Basic or Digest authentication.

http_proxy_password MobiLink Client Network Protocol Option [page 47]
: Authenticate to third-party HTTP proxies using RFC 2617 Basic or Digest authentication.

http_proxy_userid MobiLink Client Network Protocol Option [page 48]
: Authenticate to third-party HTTP proxies using RFC 2617 Basic or Digest authentication.

http_userid MobiLink Client Network Protocol Option [page 50]
: Authenticate to third-party HTTP servers and gateways using RFC 2617 Basic or Digest authentication.

identity MobiLink Client Network Protocol Option [page 51]

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

identity_password MobiLink Client Network Protocol Option [page 52]
: The password used to encrypt the private key found in the identity file.

min_tls_version MobiLink Client Network Protocol Option [page 53]
: The minimum downgrade TLS version.

network_adapter_name MobiLink Client Network Protocol Option [page 53]
: Allows MobiLink clients to explicitly specify the name of the network adapter to use to connect to MobiLink.

network_leave_open MobiLink Client Network Protocol Option [page 54]
: When you specify network_name, you can optionally specify that the network connectivity should be left open after the synchronization finishes.

network_name MobiLink Client Network Protocol Option [page 55]
: Specify the network name to start if an attempt to connect to the network fails.

persistent MobiLink Client Network Protocol Option [page 56]
: Use a single TCP/IP connection for all HTTP requests in a synchronization.

port MobiLink Client Network Protocol Option [page 57]
: Specify the socket port number of the MobiLink server.

proxy_host MobiLink Client Network Protocol Option [page 58]
: Specify the host name or IP address of the proxy server.

proxy_port MobiLink Client Network Protocol Option [page 59]
: Specify the port number of the proxy server.

set_cookie MobiLink Client Network Protocol Option [page 60]
: Specify custom HTTP cookies to set in the HTTP requests used during synchronization.

skip_certificate_name_check MobiLink Network Protocol Option [page 62]
: Controls whether the client library skips the check of the server host name against the database server certificate host names.

timeout MobiLink Client Network Protocol Option [page 63]
: Specify the amount of time, in seconds, that the client waits for network operations to succeed before giving up.

trusted_certificates MobiLink Client Network Protocol Option [page 64]
: Specify a file containing a list of trusted root certificates used for secure synchronization.

trusted_certificate_name MobiLink Client Network Protocol Option [page 66]
: Specify a unique name for a certificate stored in the remote database. This option is only available for SQL Anywhere clients.

url_suffix MobiLink Client Network Protocol Option [page 68]
: Specify the suffix to add to the URL on the first line of each HTTP request sent during synchronization.

version MobiLink Client Network Protocol Option [page 69]
: Specify the version of HTTP to use for synchronization.

zlib_download_window_size MobiLink Client Network Protocol Option [page 70]
: If you set the compression option to zlib, you can use this option to specify the compression window size for download.

zlib_upload_window_size MobiLink Client Network Protocol Option [page 71]
If you set the compression option to zlib, you can use this option to specify the compression window size for upload.

Related Information

The Relay Server Configuration File
Starting the MobiLink Profiler (Administration Tools)
CommunicationAddress (adr) Extended Option [page 166]
Stream Parameters Synchronization Parameter
UltraLite Synchronization Utility (ulsync)
MobiLink File Transfer Utility (mlfiletransfer) [page 22]
MobiLink Listener Utility for Windows (dblsn)

1.4.1 Protocol Options

The network protocol you choose for the synchronization server must match the synchronization protocol used by the client. Connection options for the MobiLink server are set using the -x mlsrv17 option.

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...
client_port= `nnnnn`[-`mmmmm`]	client_port
compression={*zlib*|*none*}	compression
e2ee_public_key= `file`	e2ee_public_key
host= `hostname`	host
network_adapter_name= `name`	network_adapter_name
network_leave_open={*off*|*on*}	network_leave_open
network_name= `name`	network_name
port= `portnumber`	port
timeout= `seconds`	timeout
zlib_download_window_size= `window-bits`	zlib_download_window_size
zlib_upload_window_size= `window-bits`	zlib_upload_window_size

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...
certificate_company= `company_name`	certificate_company
certificate_name= `name`	certificate_name
certificate_unit= `company_unit`	certificate_unit
client_port= `nnnnn`[-mmmmm]	client_port
compression={*zlib*|*none*}	compression
e2ee_public_key= `file`	e2ee_public_key
fips={*y*|*n*}	fips
host= `hostname`	host
identity= `filename`	identity
identity_name= `name`	identity_name
identity_password= `password`	identity_password
min_tls_version= `ver`	min_tls_version
network_adapter_name= `name`	network_adapter_name
network_leave_open={*off*|*on*}	network_leave_open
network_name= `name`	network_name
port= `portnumber`	port
skip_certificate_name_check={*off*|*on*}	skip_certificate_name_check
timeout= `seconds`	timeout
trusted_certificates= `filename`	trusted_certificate
trusted_certificate_name= `name`	trusted_certificate_name
zlib_download_window_size= `window-bits`	zlib_download_window_size
zlib_upload_window_size= `window-bits`	zlib_upload_window_size

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
client_port= `nnnnn`[- mmmmm]	client_port
compression={*zlib*|*none*}	compression
custom_header= `header`	custom_header
e2ee_public_key= `file`	e2ee_public_key
http_buffer_responses={*on*|*off*}	http_buffer_responses

HTTP protocol option	For more information, see...
http_password= password	http_password
http_proxy_password= password	http_proxy_password
http_proxy_userid= userid	http_proxy_userid
http_userid= userid	http_userid
host= hostname	host
network_adapter_name= name	network_adapter_name
network_leave_open={*off*|*on*}	network_leave_open
network_name= name	network_name
persistent={*off*|*on*}	persistent
port= portnumber	port
proxy_host= proxy-hostname-or-ip	proxy_host
proxy_port= proxy-portnumber	proxy_port
set_cookie= cookie-name = cookie-value	set_cookie
timeout= seconds	timeout
url_suffix= suffix	url_suffix
version= HTTP-version-number	version
zlib_download_window_size= window-bits	zlib_download_window_size
zlib_upload_window_size= window-bits	zlib_upload_window_size

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
certificate_company= company_name	certificate_company
certificate_name= name	certificate_name
certificate_unit= company_unit	certificate_unit
client_port= nnnnn[- mmmmm]	client_port
compression={*zlib*|*none*}	compression
custom_header= header	custom_header
e2ee_public_key= file	e2ee_public_key
fips={*y*|*n*}	fips
host= hostname	host
http_buffer_responses{*off* |*on* }	http_buffer_responses

HTTPS protocol option	For more information, see...
http_password= `password`	http_password
http_proxy_password= `password`	http_proxy_password
http_proxy_userid= `userid`	http_proxy_userid
http_userid= `userid`	http_userid
identity= `filename`	identity
identity_name= `name`	identity_name
identity_password= `password`	identity_password
min_tls_version= `ver`	min_tls_version
network_adapter_name= `name`	network_adapter_name
network_leave_open={*off*|*on*}	network_leave_open
network_name= `name`	network_name
persistent={*off*|*on*}	persistent
port= `portnumber`	port
proxy_host= `proxy-hostname-or-ip`	proxy_host
proxy_port= `proxy-portnumber`	proxy_port
set_cookie= `cookie-name` = `cookie-value`	set_cookie
skip_certificate_name_check={*off*|*on*}	skip_certificate_name_check
timeout= `seconds`	timeout
trusted_certificates= `filename`	trusted_certificate
trusted_certificate_name= `name`	trusted_certificate_name
url_suffix= `suffix`	url_suffix
version= `HTTP-version-number`	version
zlib_download_window_size= `window-size`	zlib_download_window_size
zlib_upload_window_size= `window-bits`	zlib_upload_window_size

Related Information

-x mlsrv17 Option

1.4.2 allow_unencrypted_basic_proxy_auth MobiLink Client Network Protocol Option

Permit Basic HTTP proxy authentication.

Syntax

```
allow_unencrypted_basic_proxy_auth={ y | n }
```

Available protocols

HTTP, HTTPS

Default

The default is no.

Remarks

By default, Basic HTTP authentication to an HTTP proxy is disabled to prevent an attack that downgrades the authentication type from Digest to Basic proxy authentication.

To perform Basic proxy authentication, specify the following protocol option:

```
allow_unencrypted_basic_proxy_auth=1
```

1.4.3 buffer_size MobiLink Client Network Protocol Option

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

Syntax

```
buffer_size=bytes
```

Available protocols

- HTTP, HTTPS

Default

- Microsoft Windows Mobile - 16 KB
- Android, iOS, Linux ARM, Microsoft Windows Phone/Store - 64 KB
- Desktop - 256 KB

Remarks

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

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.

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 = "buffer_size=32K";
```

Related Information

UltraLite Network Protocol Options
CommunicationAddress (adr) Extended Option [page 166]

1.4.4 certificate_company MobiLink Client Network Protocol Option

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

Syntax

```
certificate_company=organization
```

Available protocols

- TLS, HTTPS

Default

None

Remarks

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

When initiating TLS or HTTPS connections, the client libraries will check the host name of the database server against the certificate provided by that server using the procedure described in RFC 2818. This check will only happen if none of the certificate_name, certificate_company, or certificate_unit options are specified, and the skip_certificate_name_check option is not enabled. If any of certificate_name, certificate_company, or certificate_unit are specified, only those options are verified. The skip_certificate_name_check option disables the host name check when enabled.

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:

```
mlsrv17
   -c "DSN=SQL Anywhere 17 Demo;UID=DBA;PWD=sql"
   -x https(
     identity=myserver.id;
     identity_password=pwd)
```

On a SQL Anywhere client, the implementation is:

```
dbmlsync
   -c "DSN=mydb;UID=DBA;PWD=passwd"
   -e "ctp=https;
       adr='host=myserver=c:\certs\rootca.crt;
          certificate_company=My Company;
          certificate_unit=My Division;
          certificate_name=My MobiLink Server'"
```

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

```
info.stream = "https";
    info.stream_parms =
       "trusted_certificates=c:\cert\rootca.crt;"
          "certificate_company=My Company;"
          "certificate_unit=My Division;"
          "certificate_name=My MobiLink Server";
```

Related Information

MobiLink Client/Server Communications Encryption
Server Authentication
UltraLite Network Protocol Options
-x mlsrv17 Option
-e dbmlsync Option [page 134]
trusted_certificates MobiLink Client Network Protocol Option [page 64]
certificate_name MobiLink Client Network Protocol Option [page 34]
certificate_unit MobiLink Client Network Protocol Option [page 36]
CommunicationAddress (adr) Extended Option [page 166]
skip_certificate_name_check MobiLink Network Protocol Option [page 62]

1.4.5 certificate_name MobiLink Client Network Protocol Option

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

Syntax

```
certificate_name=common-name
```

Available protocols

- TLS, HTTPS

Default

None

Remarks

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

When initiating TLS or HTTPS connections, the client libraries will check the host name of the database server against the certificate provided by that server using the procedure described in RFC 2818. This check will only happen if none of the certificate_name, certificate_company, or certificate_unit options are specified, and the skip_certificate_name_check option is not enabled. If any of certificate_name, certificate_company, or certificate_unit are specified, only those options are verified. The skip_certificate_name_check option disables the host name check when enabled.

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:

```
mlsrv17
   -c "DSN=SQL Anywhere 17 Demo;UID=DBA;PWD=sql"
   -x https(
     identity=myserver.id;
     identity_password=pwd)
```

On a SQL Anywhere client, the implementation is:

```
dbmlsync
   -c "DSN=mydb;UID=DBA;PWD=passwd"
   -e "ctp=https;
       adr='host=MyServer=c:\certs\rootca.crt;
          certificate_company=My Company;
          certificate_unit=My Division;
          certificate_name=My MobiLink Server'"
```

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

```
info.stream = "https";
    info.stream_parms =
       "trusted_certificates=c:\cert\rootca.crt;"
          "certificate_company=My Company;"
          "certificate_unit=My Division;"
          "certificate_name=My MobiLink Server";
```

Related Information

MobiLink Client/Server Communications Encryption
Server Authentication
UltraLite Network Protocol Options
-x mlsrv17 Option
-e dbmlsync Option [page 134]
trusted_certificates MobiLink Client Network Protocol Option [page 64]
certificate_company MobiLink Client Network Protocol Option [page 33]
certificate_unit MobiLink Client Network Protocol Option [page 36]
CommunicationAddress (adr) Extended Option [page 166]
skip_certificate_name_check MobiLink Network Protocol Option [page 62]

1.4.6 certificate_unit MobiLink Client Network Protocol Option

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

Syntax

```
certificate_unit=organization-unit
```

Available protocols

- TLS, HTTPS

Default

None

Remarks

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

When initiating TLS or HTTPS connections, the client libraries will check the host name of the database server against the certificate provided by that server using the procedure described in RFC 2818. This check will only happen if none of the certificate_name, certificate_company, or certificate_unit options are specified, and the skip_certificate_name_check option is not enabled. If any of certificate_name, certificate_company, or certificate_unit are specified, only those options are verified. The skip_certificate_name_check option disables the host name check when enabled.

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:

```
mlsrv17
   -c "DSN=SQL Anywhere 17 Demo;UID=DBA;PWD=sql"
   -x https(
     identity=myserver.id;
     identity_password=pwd)
```

On a SQL Anywhere client, the implementation is:

```
dbmlsync
   -c "DSN=mydb;UID=DBA;PWD=passwd"
   -e "ctp=https;
       adr='host=MyServer=c:\certs\rootca.crt;
          certificate_company=My Company;
          certificate_unit=My Division;
          certificate_name=My MobiLink Server'"
```

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

```
info.stream = "https";
    info.stream_parms =
       "trusted_certificates=c:\cert\rootca.crt;"
          "certificate_company=My Company;"
          "certificate_unit=My Division;"
          "certificate_name=My MobiLink Server";
```

Related Information

MobiLink Client/Server Communications Encryption

Server Authentication
UltraLite Network Protocol Options
-x mlsrv17 Option
-e dbmlsync Option [page 134]
trusted_certificates MobiLink Client Network Protocol Option [page 64]
certificate_company MobiLink Client Network Protocol Option [page 33]
certificate_name MobiLink Client Network Protocol Option [page 34]
CommunicationAddress (adr) Extended Option [page 166]
skip_certificate_name_check MobiLink Network Protocol Option [page 62]

1.4.7 client_port MobiLink Client Network Protocol Option

Specify a range of client ports for communication.

Syntax

```
client_port=nnnnn [-mmmmm]
```

Available protocols

- TCPIP, TLS, HTTP, HTTPS

Default

None

Remarks

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

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

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 = "client_port=10000-19999";
```

Related Information

UltraLite Network Protocol Options
CommunicationAddress (adr) Extended Option [page 166]

1.4.8 compression MobiLink Client Network Protocol Option

Sets the type of data compression to be used.

Syntax

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

Available protocols

- TCPIP, TLS, HTTP, HTTPS

Default

For UltraLite, no compression is used by default.

For dbmlsync, zlib compression is used by default.

⚠ Caution

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

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.

When linking your application directly against the static UltraLite runtime, call `ULEnableZlibSyncCompression( sqlca )` to link the zlib functionality directly into your application. Otherwise, `mlczlib`*17*`.dll` must be deployed.

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"
```

Related Information

Transport Layer Security
UltraLite Network Protocol Options
zlib_download_window_size MobiLink Client Network Protocol Option [page 70]
zlib_upload_window_size MobiLink Client Network Protocol Option [page 71]
CommunicationAddress (adr) Extended Option [page 166]

1.4.9 custom_header MobiLink Client Network Protocol Option

Specify a custom HTTP header.

Syntax

```
custom_header=header
```

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

Available protocols

- HTTP, HTTPS

Default

None

Remarks

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

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

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 =
  "host=www.myhost.com;proxy_host=www.myproxy.com;
  custom_header=MyProxyHdr:ProxyUser";
```

Related Information

UltraLite Network Protocol Options

CommunicationAddress (adr) Extended Option [page 166]

1.4.10 e2ee_public_key MobiLink Client Network Protocol Option

Specify the file containing the server's public key or X.509 certificate for end-to-end encryption.

Syntax

e2ee_public_key=`file`

Available protocols

TCPIP, TLS, HTTP, HTTPS

Default

None

Remarks

The file can be either PEM or DER encoded.

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.

Example

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

```
info.stream = "https";
info.stream_parms =
"trusted_certificates\=rsaroot.crt;e2ee_public_key=rsapublic.pem"
```

Related Information

UltraLite Network Protocol Options
fips MobiLink Client Network Protocol Option [page 42]
-x mlsrv17 Option
Key Pair Generator Utility (createkey)
CommunicationAddress (adr) Extended Option [page 166]

1.4.11 fips MobiLink Client Network Protocol Option

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

Syntax

```
fips={ y | n }
```

Available protocols

HTTPS, TLS

Default

No

Remarks

Clients with the fips option enabled can connect to servers with the fips option disabled 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 encryption.

FIPS-certified encryption requires a separate license.

Example

The following example sets up FIPS-certified 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:

```
mlsrv17
  -c "DSN=SQL Anywhere 17 Demo;UID=DBA;PWD=sql"
  -x tls(
    fips=y;
    identity=%SQLANYSAMP17%\Certificates\rsaserver.id;
    identity_password=test)
```

On a SQL Anywhere client, the implementation is:

```
dbmlsync -e
   "CommunicationType=tls;
    CommunicationAddress=
       'fips=y;
        trusted_certificates=%SQLANYSAMP17%\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 =
       "fips=y;"
       "trusted_certificates=C:\\Users\\Public\\Documents\\SQL Anywhere
          17\\Samples\\Certificates\\rsaroot.crt;"
       "certificate_name=RSA Server;"
       "certificate_company=SAP;"
```

```
        "certificate_unit=SQL Anywhere";
```

Related Information

UltraLite Network Protocol Options
Separately Licensed Components
e2ee_public_key MobiLink Client Network Protocol Option [page 41]
CommunicationAddress (adr) Extended Option [page 166]

1.4.12 host MobiLink Client Network Protocol Option

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

Syntax

```
host=hostname-or-ip
```

Available protocols

- TCPIP, TLS, HTTP, HTTPS

Default

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

Remarks

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

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 = "host=myhost;port=1234";
```

Related Information

UltraLite Network Protocol Options
CommunicationAddress (adr) Extended Option [page 166]

1.4.13 http_buffer_responses MobiLink Client Network Protocol Option

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

Syntax

```
http_buffer_responses={ off | on }
```

Available protocols

- HTTP, HTTPS

Default

Off

Remarks

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

Related Information

UltraLite Network Protocol Options
CommunicationAddress (adr) Extended Option [page 166]

1.4.14 http_password MobiLink Client Network Protocol Option

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

Syntax

```
http_password=password
```

Available protocols

- HTTP, HTTPS

Default

None

Remarks

This feature supports Basic and Digest authentication 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.

Example

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

```
synch_info.stream = "https";
synch_info.stream_parms = "http_userid=user;http_password=passwd";
```

Related Information

UltraLite Network Protocol Options
http_userid MobiLink Client Network Protocol Option [page 50]
http_proxy_password MobiLink Client Network Protocol Option [page 47]
http_proxy_userid MobiLink Client Network Protocol Option [page 48]
CommunicationAddress (adr) Extended Option [page 166]

1.4.15 http_proxy_password MobiLink Client Network Protocol Option

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

Syntax

http_proxy_password=`password`

Available protocols

- HTTP, HTTPS

Default

None

Remarks

This feature supports Basic and Digest authentication 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.

Example

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

```
synch_info.stream = "https";
synch_info.stream_parms = "http_proxy_userid=user;http_proxy_password=passwd";
```

Related Information

UltraLite Network Protocol Options
http_password MobiLink Client Network Protocol Option [page 46]
http_userid MobiLink Client Network Protocol Option [page 50]
http_proxy_userid MobiLink Client Network Protocol Option [page 48]
CommunicationAddress (adr) Extended Option [page 166]

1.4.16 http_proxy_userid MobiLink Client Network Protocol Option

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

Syntax

http_proxy_userid=`userid`

Available protocols

- HTTP, HTTPS

Default

None

Remarks

This feature supports Basic and Digest authentication 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.

Example

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

```
synch_info.stream = "https";
synch_info.stream_parms = "http_proxy_userid=user;http_proxy_password=passwd";
```

Related Information

UltraLite Network Protocol Options
http_password MobiLink Client Network Protocol Option [page 46]
http_userid MobiLink Client Network Protocol Option [page 50]
http_proxy_password MobiLink Client Network Protocol Option [page 47]
CommunicationAddress (adr) Extended Option [page 166]

1.4.17 http_userid MobiLink Client Network Protocol Option

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

Syntax

```
http_userid=userid
```

Available protocols

- HTTP, HTTPS

Default

None

Remarks

This feature supports Basic and Digest authentication 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.

Example

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

```
synch_info.stream = "https";
synch_info.stream_parms = "http_userid=user;http_password=passwd";
```

Related Information

UltraLite Network Protocol Options

http_password MobiLink Client Network Protocol Option [page 46]
http_proxy_password MobiLink Client Network Protocol Option [page 47]
http_proxy_userid MobiLink Client Network Protocol Option [page 48]
CommunicationAddress (adr) Extended Option [page 166]

1.4.18 identity MobiLink Client Network Protocol Option

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

Syntax

```
identity=filename
```

Available protocols

TLS, HTTPS

Default

None

Remarks

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

If the private key is encrypted, use the identity_password option to specify a password.

Client-side certificates allow you to authenticate to third-party servers and proxies that have been configured to accept client-side certificate authentication and are sitting between the client and MobiLink server. Client-side certificates can also provide client-side authentication with the MobiLink server by retrieving the client-side certificates from within an authentication_user connection event by using the getCertificateChain method and testing its authenticity.

Related Information

UltraLite Network Protocol Options

identity_password MobiLink Client Network Protocol Option [page 52]
CommunicationAddress (adr) Extended Option [page 166]

1.4.19 identity_password MobiLink Client Network Protocol Option

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

Syntax

```
identity_password=password
```

Available protocols

- TLS, HTTPS

Default

None.

Remarks

This option is only required if the private key in the identity file is encrypted.

Related Information

UltraLite Network Protocol Options
identity MobiLink Client Network Protocol Option [page 51]
CommunicationAddress (adr) Extended Option [page 166]

1.4.20 min_tls_version MobiLink Client Network Protocol Option

The minimum downgrade TLS version.

Syntax

```
min_tls_version=ver
```

Available protocols

- TLS, HTTPS

Default

1.2

Remarks

The supported values for `ver` are *1.0*, *1.1*, and *1.2* (the default). The period is optional, so you can also specify *10*, *11*, or *12*.

1.4.21 network_adapter_name MobiLink Client Network Protocol Option

Allows MobiLink clients to explicitly specify the name of the network adapter to use to connect to MobiLink.

Syntax

```
network_adapter_name=name
```

Available protocols

- TCPIP, TLS, HTTP, HTTPS

Default

None

Remarks

This option is valid only for Windows desktop.

Related Information

UltraLite Network Protocol Options
CommunicationAddress (adr) Extended Option [page 166]

1.4.22 network_leave_open MobiLink Client Network Protocol Option

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

Syntax

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

Available protocols

- TCPIP, TLS, HTTP, HTTPS

Default

The default is *off*.

Remarks

You must specify network_name to use this option.

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

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 = "network_name=MyNetwork;network_leave_open=on";
```

Related Information

UltraLite Network Protocol Options
network_name MobiLink Client Network Protocol Option [page 55]
CommunicationAddress (adr) Extended Option [page 166]

1.4.23 network_name MobiLink Client Network Protocol Option

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

Syntax

```
network_name=name
```

Available protocols

- TCPIP, TLS, HTTP, HTTPS

Default

None

Remarks

Specify the network name so that you can use the MobiLink auto-dial feature. This allows you to connect from a Microsoft Windows Mobile device or Microsoft 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 database can synchronize unattended. When used without scheduling, this allows you to run dbmlsync without manually dialing a connection.

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

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 = "network_name=MyNetwork;network_leave_open=on";
```

Related Information

Synchronization Schedules [page 110]
UltraLite Network Protocol Options
network_leave_open MobiLink Client Network Protocol Option [page 54]
CommunicationAddress (adr) Extended Option [page 166]

1.4.24 persistent MobiLink Client Network Protocol Option

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

Syntax

```
persistent={ off | on }
```

Available protocols

- HTTP, HTTPS

Default

On

Remarks

The On value means that the client attempts to use the same TCP/IP connection for all HTTP requests in a synchronization.

If an intermediary server requests non-persistent connections or the client detects that an intermediary does not support persistent connections, the connection is automatically downgraded to non-persistent for the rest of the synchronization session.

Related Information

UltraLite Network Protocol Options
CommunicationAddress (adr) Extended Option [page 166]

1.4.25 port MobiLink Client Network Protocol Option

Specify the socket port number of the MobiLink server.

Syntax

```
port=port-number
```

Available 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.

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 = "host=myhost;port=1234";
```

Related Information

UltraLite Network Protocol Options
CommunicationAddress (adr) Extended Option [page 166]

1.4.26 proxy_host MobiLink Client Network Protocol Option

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

Syntax

```
proxy_host=proxy-hostname-or-ip
```

Available protocols

- HTTP, HTTPS

Default

None

Remarks

Use only if going through an HTTP proxy.

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 = "proxy_host=myproxyhost;proxy_port=1234";
```

Related Information

UltraLite Network Protocol Options
CommunicationAddress (adr) Extended Option [page 166]

1.4.27 proxy_port MobiLink Client Network Protocol Option

Specify the port number of the proxy server.

Syntax

```
proxy_port=proxy-port-number
```

Available protocols

- HTTP, HTTPS

Default

None

Remarks

Use only if going through an HTTP proxy.

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 = "proxy_host=myproxyhost;proxy_port=1234";
```

Related Information

UltraLite Network Protocol Options
CommunicationAddress (adr) Extended Option [page 166]

1.4.28 set_cookie MobiLink Client Network Protocol Option

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

Syntax

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

Available protocols

- HTTP, HTTPS

Default

None

Remarks

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

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

Example

The following example sets two custom HTTP cookies in an Embedded SQL or C++ UltraLite application.

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

Related Information

UltraLite Network Protocol Options
CommunicationAddress (adr) Extended Option [page 166]

1.4.29 skip_certificate_name_check MobiLink Network Protocol Option

Controls whether the client library skips the check of the server host name against the database server certificate host names.

Available protocols

HTTPS, TLS

Default

None

Remarks

Set this boolean option to ON or OFF to control whether the host name of the database server matches any of the host names in the server's certificate. Note, however, that enabling this option may prevent the client from fully authenticating the server, leaving it vulnerable to attack.

When initiating TLS or HTTPS connections, the client libraries will check the host name of the database server against the certificate provided by that server. This check will only happen if none of the certificate_name, certificate_company, or certificate_unit options are specified, or the skip_certificate_name_check option is not enabled. If any of certificate_name, certificate_company, or certificate_unit are specified, only those options are verified. The skip_certificate_name_check option disables the host name check when enabled.

The client matches the host name of the server against the host names listed in the subjectAltName (Subject Alternative Name or SAN) extension and the Common Name (CN) field if the SAN is not present. The SAN may contain multiple host names with wild cards. For example, a Google certificate might include *.google.com, *.google.ca, and *.android.com. Thus, www.google.ca is a valid host name.

HTTPS is only supported for web services client procedures.

Example

The following connection string fragment verifies that myrootcert.crt is at the root of the database server's certificate's signing chain but does not verify that myhost.com is identified in the certificate.

```
HOST=myhost.com;SERVER=myserver;ENCRYPTION=TLS(trusted_certificate=myrootcert.crt
;skip_certificate_name_check=ON)
```

Related Information

MobiLink Client/Server Communications Encryption
Server Authentication
UltraLite Network Protocol Options
-x mlsrv17 Option
-e dbmlsync Option [page 134]
trusted_certificates MobiLink Client Network Protocol Option [page 64]
certificate_company MobiLink Client Network Protocol Option [page 33]
certificate_name MobiLink Client Network Protocol Option [page 34]
certificate_unit MobiLink Client Network Protocol Option [page 36]
CommunicationAddress (adr) Extended Option [page 166]

1.4.30 timeout MobiLink Client Network Protocol Option

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

Syntax

```
timeout=seconds
```

Available protocols

- TCPIP, TLS, HTTP, HTTPS

Default

240 seconds

Remarks

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

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

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

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

The value 0 means that the timeout is 10 minutes.

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 = "timeout=300";
```

Related Information

UltraLite Network Protocol Options
CommunicationAddress (adr) Extended Option [page 166]

1.4.31 trusted_certificates MobiLink Client Network Protocol Option

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

Syntax

```
trusted_certificates=filename
```

Available protocols

- TLS, HTTPS

Default

None

Remarks

The trusted_certificates parameter takes the name of a file that contains a list of PEM-encoded X.509 trusted root certificates.

When synchronization occurs through a 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.

Certificates are used according to the following rules of precedence:

- For UltraLite clients, if certificates were set in the database by ulinit or ulload, then those certificates are used.
- If the trusted_certificates parameter is provided, then the certificates from the specified file are used, replacing any trusted certificates that were stored in the database using ulinit or ulload.
- If certificates are not specified by either the trusted_certificates parameter or by ulinit or ulload and you are on Microsoft Windows, Microsoft Windows Mobile, or Android, certificates are read 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.

You cannot use both the trusted_certificates option and the trusted_certificate_name option in the same set of stream options.

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:

```
mlsrv17
   -c "DSN=SQL Anywhere 17 Demo;UID=DBA;PWD=sql"
   -x https(
     identity=%SQLANYSAMP17%\Certificates\rsaserver.id;
     identity_password=test)
```

On a SQL Anywhere client, the implementation is:

```
dbmlsync
   -c "DSN=mydb;UID=DBA;PWD=passwd"
   -e "ctp=https;
       adr='trusted_certificates=%SQLANYSAMP17%\Certificates\rsaroot.crt;
          certificate_name=RSA Server;
          certificate_company=SAP;
          certificate_unit=SQL Anywhere'"
```

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

```
info.stream = "https";
    info.stream_parms =
       "trusted_certificates=C:\\Users\\Public\\Documents\\SQL Anywhere
          17\\Samples\\Certificates\\rsaroot.crt;"
       "certificate_name=RSA Server;"
       "certificate_company=SAP;"
       "certificate_unit=SQL Anywhere";
```

Related Information

UltraLite File Path Formats in Connection Parameters
MobiLink Client/Server Communications Encryption
UltraLite Network Protocol Options
trusted_certificate_name MobiLink Client Network Protocol Option [page 66]
-x mlsrv17 Option
certificate_company MobiLink Client Network Protocol Option [page 33]
certificate_name MobiLink Client Network Protocol Option [page 34]
certificate_unit MobiLink Client Network Protocol Option [page 36]
CommunicationAddress (adr) Extended Option [page 166]

1.4.32 trusted_certificate_name MobiLink Client Network Protocol Option

Specify a unique name for a certificate stored in the remote database. This option is only available for SQL Anywhere clients.

Syntax

trusted_certificate_name=`name`

Available protocols

- TLS, HTTPS

Default

None

Remarks

When this option is specified, dbmlsync retrieves the specified certificate from the database and treats it as a trusted certificate for the purpose of validating the identity of the server.

You cannot use both the trusted_certificate_name option and the trusted_certificates option in the same set of stream options.

Example

In the following example, assume `root.crt` contains a certificate that you want to use as a trusted root certificate. To insert the certificate into the database, use the following syntax:

```
CREATE CERTIFICATE myroot FROM FILE 'root.crt'
```

Then use the following dbmlsync options to access the certificate.

```
ALTER SYNCHRONIZATION SUBSCRIPTION mysub
TYPE tls
ADDRESS 'trusted_certificate_name=myroot;...'
```

Related Information

UltraLite File Path Formats in Connection Parameters
MobiLink Client/Server Communications Encryption
UltraLite Network Protocol Options

trusted_certificates MobiLink Client Network Protocol Option [page 64]

-x mlsrv17 Option

certificate_company MobiLink Client Network Protocol Option [page 33]

certificate_name MobiLink Client Network Protocol Option [page 34]

certificate_unit MobiLink Client Network Protocol Option [page 36]

CommunicationAddress (adr) Extended Option [page 166]

1.4.33 url_suffix MobiLink Client Network Protocol Option

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

Syntax

```
url_suffix=suffix
```

The syntax of `suffix` depends on whether you are using the IIS or Apache Relay Server:

Redirector	Syntax of `suffix`
IIS	The default for Windows is `/rs/client/rs.dll`.
Apache	The default for Linux is `/cli/iarelayserver`.

Available protocols

- HTTP, HTTPS

Default

The default is `/`.

Remarks

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

For information about how to set this option when using the Relay Server, see the Relay Server configuration file documentation.

Related Information

The Relay Server Configuration File
UltraLite Network Protocol Options
CommunicationAddress (adr) Extended Option [page 166]

1.4.34 version MobiLink Client Network Protocol Option

Specify the version of HTTP to use for synchronization.

Syntax

```
version=HTTP-version-number
```

Available protocols

- HTTP, HTTPS

Default

The default value is *1.1*.

Remarks

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

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 = "version=1.0";
```

Related Information

UltraLite Network Protocol Options
CommunicationAddress (adr) Extended Option [page 166]

1.4.35 zlib_download_window_size MobiLink Client Network Protocol Option

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

Syntax

```
zlib_download_window_size=window-bits
```

Available protocols

- TCPIP, TLS, HTTP, HTTPS

Default

12 on mobile operating systems, 15 on desktop operating systems.

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 = $2^{(\text{window-bits})}$

When linking your application directly against the static UltraLite runtime, call `ULEnableZlibSyncCompression( sqlca )` to link the zlib functionality directly into your application. Otherwise, `mlczlib`*17*`.dll` must be deployed.

Example

The following option sets compression for upload only:

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

Related Information

UltraLite Network Protocol Options
compression MobiLink Client Network Protocol Option [page 39]
zlib_upload_window_size MobiLink Client Network Protocol Option [page 71]
CommunicationAddress (adr) Extended Option [page 166]

1.4.36 zlib_upload_window_size MobiLink Client Network Protocol Option

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

Syntax

```
zlib_upload_window_size=window-bits
```

Available protocols

- TCPIP, TLS, HTTP, HTTPS

Default

12 on mobile operating systems, 15 on desktop operating systems.

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 = $2^{(\text{window-size})}$

When linking your application directly against the static UltraLite runtime, call `ULEnableZlibSyncCompression( sqlca )` to link the zlib functionality directly into your application. Otherwise, `mlczlib`*17*`.dll` must be deployed.

Example

The following option sets compression for download only:

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

Related Information

UltraLite Network Protocol Options
compression MobiLink Client Network Protocol Option [page 39]
zlib_download_window_size MobiLink Client Network Protocol Option [page 70]
CommunicationAddress (adr) Extended Option [page 166]

1.5 Schema Changes in Remote MobiLink Clients

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

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

The SQL syntax to support this feature is as follows:

CREATE SYNCHRONIZATION SUBSCRIPTION statement

Use the SCRIPT VERSION clause to specify the script version to use during synchronization.

ALTER SYNCHRONIZATION SUBSCRIPTION statement

Use the SET SCRIPT VERSION clause to specify the script version to use during synchronization.

In this section:

You can change a table definition in a SQL Anywhere remote database that has been deployed.

Schema Upgrades for UltraLite Remote Databases [page 77]
You can change the schema of a remote UltraLite database by having your existing application execute DDL.

Related Information

CREATE SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink]
ALTER SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink]

1.5.1 Script Versions and Subscriptions

Starting in version 12.0.0, new functionality greatly simplifies the process of performing schema changes to remote databases.

To use this functionality, you must stop using the dbmlsync ScriptVersion extended option. Instead, you should associate your script version directly with your synchronization subscription using new clauses that have been added to the CREATE SYNCHRONIZATION SUBSCRIPTION and ALTER SYNCHRONIZATION SUBSCRIPTION statements.

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

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

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

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

Example

Associate a script version with a subscription

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

1. Determine the script version currently being used to synchronize `my_sub`. The easiest way to do this is as follows:
    1. Add the -v+ option to your existing dbmlsync command line and synchronize.

    2. Look in your dbmlsync output file for a line that identifies the script version being used. Look for something similar to the following:

    ```
    Script version: my_script_ver_1
    ```

2. Associate the current script version with the subscription:

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

   where <sub_name> is the name of the subscription, in this case **`my_sub`** and <ver> is the current script version, determined in step 1.
   All transactions that occur after this point are associated with the script version.
3. Synchronize one last time using your old options. This ensures that any transactions that occurred before you completed step 2 are uploaded with the correct script version.
4. Remove the ScriptVersion extended option wherever it is specified for this subscription. The extended option may be specified on the dbmlsync command line, in a synchronization profile or associated with a subscription, publication or MobiLink user in the remote database.

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

1.5.2 Adding Tables to Deployed SQL Anywhere Remote Databases

You can add tables to remote SQL Anywhere databases after they are deployed.

Prerequisites

You must be the owner of the publication, or have one of the following:

- ALTER privilege on the publication
- SYS_REPLICATION_ADMIN_ROLE system role

Context

i Note

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

Procedure

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 an sp_hook_dbmlsync_schema_upgrade hook.
4. Synchronize.

Results

The tables are added to the remote database.

Next Steps

Use the new script version, if required.

Related Information

Script Versions
sp_hook_dbmlsync_schema_upgrade [page 283]
ALTER PUBLICATION Statement [MobiLink] [SQL Remote]
ALTER TABLE Statement

1.5.3 Altering a Published Table in a Deployed SQL Anywhere Remote Database

You can change a table definition in a SQL Anywhere remote database that has been deployed.

Prerequisites

You must be the owner of the publication, or have one of the following:

- ALTER privilege on the publication
- SYS_REPLICATION_ADMIN_ROLE system role

Context

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.

Procedure

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

   You can use this statement inside a sp_hook_dbmlsync_schema_upgrade hook.
5. At the remote database, use the ALTER TABLE statement to alter the table.
6. At the remote database, use the ALTER PUBLICATION statement to add the table back into the publication.

   You can use this statement inside an sp_hook_dbmlsync_schema_upgrade hook.
7. Synchronize with the new script version.

Results

The published table is altered.

Related Information

ALTER PUBLICATION Statement [MobiLink] [SQL Remote]
ALTER TABLE Statement
sp_hook_dbmlsync_schema_upgrade [page 283]

1.5.4 Schema Upgrades for UltraLite Remote Databases

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

When performing a schema upgrade for an UltraLite remote database, keep the following in mind:

- 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 must 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 the UltraLite SQL statements.

Related Information

UltraLite SQL Statements
Deploying UltraLite Database Schema Upgrades

1.6 SQL Anywhere Clients for MobiLink

The MobiLink server supports the use of SQL Anywhere as a MobiLink client.

In this section:

SQL Anywhere Clients [page 78]
Any SQL Anywhere database can be used as a remote database in a MobiLink system.

MobiLink SQL Anywhere Client Utility (dbmlsync) Syntax [page 115]
Use the dbmlsync utility to synchronize SQL Anywhere remote databases with a consolidated database. To run dbmlsync you must have the SYS_RUN_REPLICATION_ROLE system privilege.

MobiLink SQL Anywhere Client Extended Options [page 160]
Extended options can be specified on the dbmlsync command line using the -e or -eu options, or they can be stored in the database.

MobiLink SQL Statements [page 202]
The following are the SQL statements used for configuring and running MobiLink SQL Anywhere clients:

MobiLink Synchronization Profiles [page 203]
Synchronization profiles allow you to place some dbmlsync options in the database. The synchronization profile you create can contain a variety of synchronizations options.

Event Hooks for SQL Anywhere Clients [page 232]
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.

Dbmlsync C++ API Reference [page 301]
The Dbmlsync C++ API provides a programming interface that allows MobiLink client applications written in C++ to launch synchronizations and receive feedback about the progress of the synchronizations they request.

Dbmlsync .NET API Reference [page 302]
The Dbmlsync .NET API provides a programming interface that allows MobiLink client applications written in .NET to launch synchronizations and receive feedback about the progress of the synchronizations they request.

DBTools Interface for dbmlsync [page 303]
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.

Scripted Upload [page 308]
Scripted upload applies only to MobiLink applications that use SQL Anywhere remote databases.

1.6.1 SQL Anywhere Clients

Any SQL Anywhere database can be used as a remote database in a MobiLink system.

In this section:

Using a SQL Anywhere Database as a Remote Database [page 79]
To use a SQL Anywhere database as a remote database in a MobiLink system, you need to create a publication, create a MobiLink user, create a synchronization subscription, and register the user with the consolidated database.

Publications [page 85]

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.)

MobiLink Users [page 95]
: 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.

Synchronization Subscription Creation [page 100]
: 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.

Initiation of Synchronization [page 102]
: The client always initiates MobiLink synchronization. For SQL Anywhere clients, synchronization can be initiated using the dbmlsync utility, the dbmlsync API or the SQL SYNCHRONIZE statement. All share similar semantics but offer different interfaces to synchronization and different abilities to integrate synchronization with your own applications.

Synchronization Schedules [page 110]
: You can set up dbmlsync to synchronize periodically based on rules you define.

dbmlsync Synchronization Customization [page 112]
: You can use event hooks, dbmlsync programming interfaces and scripted upload to customize dbmlsync synchronizations.

Dbmlsync API [page 113]
: 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.

SQL Anywhere Client Logs [page 114]
: When you create MobiLink applications with SQL Anywhere remote databases, there are two types of client log files: the dbmlsync message log and the SQL Anywhere transaction log.

Version Considerations [page 115]
: In order for dbmlsync to function properly, both the major and minor versions of dbmlsync.exe must match those of the database server.

1.6.1.1 Using a SQL Anywhere Database as a Remote Database

To use a SQL Anywhere database as a remote database in a MobiLink system, you need to create a publication, create a MobiLink user, create a synchronization subscription, and register the user with the consolidated database.

Context

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.

i Note

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

Procedure

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.
3. Create MobiLink users in the remote database.
4. Register users with the consolidated database.
5. Create synchronization subscriptions in the remote database.

Results

The SQL Anywhere database is set up as a remote database.

In this section:

Deploying MobiLink Remote Databases by Customizing a Prototype [page 81]
To deploy SQL Anywhere remote databases, you need to create the databases and add the appropriate publications. To do this, you can customize a prototype remote database.

Remote ID Settings [page 82]
The remote ID uniquely identifies a remote database in a MobiLink synchronization system.

Remote Database Upgrades [page 83]
If you install a new SQL Anywhere remote database over an older version, the synchronization progress information in the consolidated database is incorrect. You can correct this problem using the ml_reset_sync_state stored procedure to reset the state information for the remote database in the consolidated database.

Progress Offsets [page 84]
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.

Related Information

Publications [page 85]
MobiLink Users [page 95]
MobiLink User Creation and Registration [page 10]
Synchronization Subscription Creation [page 100]

Scripted Upload [page 308]
Download-only Publications [page 90]

1.6.1.1.1 Deploying MobiLink Remote Databases by Customizing a Prototype

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

Context

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

When you use the *Create Synchronization Model Wizard* to create your MobiLink client application, you can deploy your database using a wizard.

Procedure

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:
   a. Create a directory to hold the remote database.
   b. Copy the prototype remote database into the directory.
   c. If the transaction log is held in the same directory as the remote database, the transaction log file name does not need to be changed.
   d. Run a SQL script that adds the individual information to the database.

      The SQL script can be a parameterized script.

Results

The remote databases are deployed.

Example

The following SQL script is taken from the Contact sample. It can be found in *%SQLANYSAMP17%*\MobiLink\Contact\customize.sql.

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

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

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

Related Information

Remote ID Settings [page 82]
Synchronization Model Deployment
Progress Offsets [page 84]
SQL Script Files
SQL Anywhere MobiLink Client Deployment
PARAMETERS Statement [Interactive SQL]

1.6.1.1.2 Remote ID Settings

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 17 plug-in to SQL Central.

The remote ID must be unique within your synchronization system.

If you set the remote ID manually and you subsequently recreate the remote database, you must either give the recreated remote database a different name from the old one or use the ml_reset_sync_state stored procedure to reset the state information in the consolidated database for the remote database.

⚠ Caution

In most cases, you do not need to set the remote ID or know its value. However, in the event that you do need to change it, 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.

Example

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

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

Related Information

Database Options
Remote IDs [page 15]
ml_remote_id Option [MobiLink]
ml_reset_sync_state System Procedure
SET OPTION Statement
SYSOPTION System View

1.6.1.1.3 Remote Database Upgrades

If you install a new SQL Anywhere remote database over an older version, the synchronization progress information in the consolidated database is incorrect. You can correct this problem using the ml_reset_sync_state stored procedure to reset the state information for the remote database in the consolidated database.

Related Information

SQL Anywhere MobiLink Client Upgrades
ml_reset_sync_state System Procedure

1.6.1.1.4 Progress Offsets

The progress offset is an integer value that indicates the point in time up to which all operations for the subscription have been uploaded and acknowledged. The dbmlsync utility uses the offset to decide what data to upload.

On the remote database, the offset is stored in the progress column of the SYS.ISYSSYNC system table. On the consolidated database, the offset is stored in the progress column of the ml_subscription table.

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

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

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

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 -s option with two subscriptions (-s sub1,pub2), and sub1 has synchronized before but sub2 has not, then the progress offsets of both subscriptions are checked against the consolidated database values.

Related Information

Transaction Log Files [page 107]

-r dbmlsync Option [page 147]

SYSSYNC System View

1.6.1.2 Publications

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 SQL Central or with the CREATE PUBLICATION statement.

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

By default, trigger actions in a SQL Anywhere client are not synchronized to the MobiLink server, on the assumption that the same triggers exist in the backend database to which the data is synchronized.

Notes about publications

- The SYS_REPLICATION_ADMIN_ROLE system role 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 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.

In this section:

Related Information

CREATE PUBLICATION Statement [MobiLink] [SQL Remote]
SendTriggers (st) Extended Option [page 190]

1.6.1.2.1 Publishing a Whole Table

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

Prerequisites

You must have the SYS_REPLICATION_ADMIN_ROLE system role.

The tables to be published must already exist.

Procedure

1. Connect to the remote database using the SQL Anywhere 17 plug-in.
2. Double-click *Publications*.
3. Click *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*.

Results

A publication is created.

Related Information

CREATE PUBLICATION Statement [MobiLink] [SQL Remote]

1.6.1.2.2 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.

Prerequisites

There is an existing remote database and you have the SYS_REPLICATION_ADMIN_ROLE system role.

Context

i Note

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

Procedure

1. Connect to the remote database using the SQL Anywhere 17 plug-in.
2. Double-click *Publications*.
3. Click *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*.

Results

The selected columns are published.

Related Information

CREATE PUBLICATION Statement [MobiLink] [SQL Remote]

1.6.1.2.3 Creating a Publication Using a WHERE Clause

When no WHERE clause is specified in an article definition, all changed rows in the table are uploaded. 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.

Prerequisites

There is an existing remote database and you have the SYS_REPLICATION_ADMIN_ROLE system role.

Context

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.

Procedure

1. Connect to the remote database using the SQL Anywhere 17 plug-in.
2. Double-click *Publications*.
3. Click *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*.

Results

The new publication is created.

Related Information

CREATE PUBLICATION Statement [MobiLink] [SQL Remote]

1.6.1.2.4 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 transaction log file.	Does not require a transaction log file.
The transaction 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 transaction log file, the synchronization does not affect the synchronization truncation point so the transaction 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.

Related Information

Upload-only and Download-only Synchronizations
CREATE PUBLICATION Statement [MobiLink] [SQL Remote]

1.6.1.2.5 Modifying the Properties of Existing Publications or Articles

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.

Prerequisites

There is an existing remote database and you have the SYS_REPLICATION_ADMIN_ROLE system role.

Publications can be altered only by the DBA or the publication's owner.

Context

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.

Procedure

1. Connect to the remote database.
2. In the left pane, click the publication or article. The properties appears in the right pane.
3. Configure the properties.

Results

The publication or article is modified.

Related Information

Schema Changes in Remote MobiLink Clients [page 72]

1.6.1.2.6 Adding Articles

Articles can be added to a publication after it has been created.

Prerequisites

There is an existing remote database and you have the SYS_REPLICATION_ADMIN_ROLE system role.

Publications can be altered only by the DBA or the publication's owner.

Procedure

1. Connect to the remote database using the SQL Anywhere 17 plug-in.
2. Double-click *Publications*.
3. Select a publication.
4. Click *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*.

Results

The specified article is added to the publication.

1.6.1.2.7 Removing Articles

An article can be deleted if it is no longer required.

Prerequisites

There is an existing remote database and you have the SYS_REPLICATION_ADMIN_ROLE system role.

Publications can be altered only by the DBA or the publication's owner.

Procedure

1. Connect to the database using the SQL Anywhere 17 plug-in.
2. Double-click *Publications*.
3. Right-click the publication and choose *Delete*.
4. Click *Yes*.

Results

The selected article is deleted.

1.6.1.2.8 Modifying an Existing Publication Using SQL

Articles and publications can be modified after they are created.

Prerequisites

There is an existing remote database and you have the SYS_REPLICATION_ADMIN_ROLE system role.

Publications can be altered only by the DBA or the publication's owner.

Procedure

1. Connect to the remote database.
2. Execute an ALTER PUBLICATION statement.

Results

The publication or article is modified.

Example

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

```
ALTER PUBLICATION pub_contact
```

```
  ADD TABLE Customers
```

Related Information

ALTER PUBLICATION Statement [MobiLink] [SQL Remote]

1.6.1.2.9 Dropping a Publication

You can drop a publication when it is no longer required.

Prerequisites

You must have a user ID that has the SYS_REPLICATION_ADMIN_ROLE system role to drop a publication, or be the owner of the publication.

Procedure

1. Connect to the remote database using the SQL Anywhere 17 plug-in.
2. Double-click *Publications*.
3. Right-click a publication and choose *Delete*.

Results

The selected publication is deleted.

Related Information

DROP PUBLICATION Statement [MobiLink] [SQL Remote]

1.6.1.3 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.

In this section:

Related Information

MobiLink User Creation and Registration [page 10]
CREATE SYNCHRONIZATION USER Statement [MobiLink]

1.6.1.3.1 Adding a MobiLink User to a Remote Database Using SQL Central

Create a MobiLink user on the remote database to be used to authenticate when you connect to the MobiLink server on the consolidated database.

Prerequisites

You must have an existing database and a user ID that has the SYS_REPLICATION_ADMIN_ROLE system role.

Procedure

1. Connect to the database using the SQL Anywhere 17 plug-in.
2. Double-click *MobiLink Users*.
3. Click *File* > *New* > *MobiLink User*.
4. In the *What Do You Want To Name The New MobiLink User* field, enter a name for the MobiLink user.
5. Click *Finish*.

Results

The MobiLink user is created.

Next Steps

Register the MobiLink user on the consolidated database.

Related Information

MobiLink User Creation and Registration [page 10]

1.6.1.3.2 Adding a MobiLink User to a Remote Database Using SQL

Create a MobiLink user on the remote database to be used to authenticate when you connect to the MobiLink server on the consolidated database.

Prerequisites

You must have an existing database and a user ID that has the SYS_REPLICATION_ADMIN_ROLE system role.

Procedure

1. Connect to the database.
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.

   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.

Results

The MobiLink user is created.

Example

The following example adds a MobiLink user named SSinger:

```
CREATE SYNCHRONIZATION USER SSinger
```

Next Steps

Register the MobiLink user on the consolidated database.

Related Information

MobiLink User Creation and Registration [page 10]
CREATE SYNCHRONIZATION USER Statement [MobiLink]
ALTER SYNCHRONIZATION USER Statement [MobiLink]

1.6.1.3.3 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.

Prerequisites

You must have an existing database and a user ID that has the SYS_REPLICATION_ADMIN_ROLE system role.

Context

You can programmatically customize the behavior of an upcoming synchronization using the sp_hook_dbmlsync_set_extended_options stored procedure.

Procedure

1. Double-click *Users*.
2. Double-click the MobiLink user name and choose *Properties*.
3. Change the properties as needed.

Results

The specified options for the MobiLink user are saved.

Related Information

MobiLink SQL Anywhere Client Extended Options [page 160]
dbmlsync Extended Options [page 106]
sp_hook_dbmlsync_set_extended_options [page 285]
ALTER SYNCHRONIZATION USER Statement [MobiLink]
CREATE SYNCHRONIZATION USER Statement [MobiLink]

1.6.1.3.4 Storing Extended Options for MobiLink Users Using SQL

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.

Prerequisites

You must have an existing database and a user ID that has the SYS_REPLICATION_ADMIN_ROLE system role.

Context

You can programmatically customize the behavior of an upcoming synchronization using the sp_hook_dbmlsync_set_extended_options extended option.

Procedure

1. Connect to the database.
2. Execute an ALTER SYNCHRONIZATION USER statement.

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

Results

The specified options for the MobiLink user are saved.

Example

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

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

Related Information

MobiLink SQL Anywhere Client Extended Options [page 160]
dbmlsync Extended Options [page 106]
sp_hook_dbmlsync_set_extended_options [page 285]
ALTER SYNCHRONIZATION USER Statement [MobiLink]
CREATE SYNCHRONIZATION USER Statement [MobiLink]

1.6.1.4 Synchronization Subscription Creation

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.

> **i 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.

Example

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

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

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

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

   ```
   CREATE SYNCHRONIZATION USER demo_ml_user
   ```

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

   ```
   CREATE SYNCHRONIZATION SUBSCRIPTION my_sub TO testpub
   ```

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

In this section:

Altering a MobiLink Synchronization Subscription [page 101]
    You can alter an existing synchronization subscriptions.

Dropping MobiLink Subscriptions [page 102]
    You can drop a synchronization subscription for a MobiLink user if it is no longer required.

Related Information

Publications [page 85]
MobiLink Users [page 95]
CREATE SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink]

1.6.1.4.1 Altering a MobiLink Synchronization Subscription

You can alter an existing synchronization subscriptions.

Prerequisites

You must have an existing database and the SYS_REPLICATION_ADMIN_ROLE system role.

Procedure

1. Connect to the database.
2. Double-click *Users*.
3. Double-click a user.
4. Right-click the subscription you want to change and select *Properties*.
5. Change the properties as needed.

Results

The synchronization subscription is modified.

Related Information

ALTER SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink]
ALTER SYNCHRONIZATION USER Statement [MobiLink]
CREATE SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink]

1.6.1.4.2 Dropping MobiLink Subscriptions

You can drop a synchronization subscription for a MobiLink user if it is no longer required.

Prerequisites

You must have the SYS_REPLICATION_ADMIN_ROLE system role to drop a synchronization subscription.

Procedure

1. Connect to the database.
2. Double-click *Users*.
3. Double-click a MobiLink user.
4. Right-click a subscription and choose *Delete*.

Results

The selected synchronization subscription is deleted.

Related Information

DROP SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink]

1.6.1.5 Initiation of Synchronization

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

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

The -c option lets you specify connection parameters that control how dbmlsync will connect to the remote database. This information is not required when using the SQL synchronize statement because the connection information is taken from the database connection that is executing the statement.

The -s or Subscription option allows you to specify which subscription defined in the remote database will be synchronized.

The CommunicationAddress and CommunicationType extended options let you specify network protocol options that determine how dbmlsync will connect with the MobiLink server during synchronization.

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

Privileges for dbmlsync

To synchronize, dbmlsync must connect to the remote database with a user ID that has the SYS_RUN_REPLICATION_ROLE system role.

Customizing synchronization

You can use event hooks, dbmlsync programming interfaces and scripted upload to customize dbmlsync synchronizations.

In this section:

Concurrency During Synchronization [page 108]
: To ensure the integrity of synchronizations, dbmlsync must ensure that no changes downloaded from the server modify rows in the remote database that have been changed since the last upload was sent.

Synchronization Initiation from an Application [page 109]
: You may want to include the features of dbmlsync in your application, rather than provide a separate executable to your remote users.

Related Information

dbmlsync Synchronization Customization [page 112]
Dbmlsync API [page 113]
MobiLink SQL Anywhere Client Utility (dbmlsync) Syntax [page 115]
SYNCHRONIZE Statement [MobiLink]
-c dbmlsync Option [page 127]
-s dbmlsync Option [page 148]
CommunicationAddress (adr) Extended Option [page 166]
CommunicationType (ctp) Extended Option [page 167]
CREATE SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink]
GRANT ROLE Statement

1.6.1.5.1 Security Considerations with Role-based Access Control and Synchronization

A user must be granted the SYS_RUN_REPLICATION_ROLE to run synchronization. The SYS_RUN_REPLICATION_ROLE grants a user system privileges, but these privileges can only be used when the user is logged in through an authenticated tool like dbmlsync or SQL Remote. This is similar to the way REMOTE DBA worked.

By default, the SYS_RUN_REPLICATION_ROLE system role includes the SYS_AUTH_DBA_ROLE compatibility role. However, the SYS_AUTH_DBA_ROLE compatibility role can be revoked from the SYS_RUN_REPLICATION_ROLE system role. The SYS_AUTH_DBA_ROLE compatibility role is the only authority that can be removed from the SYS_RUN_REPLICATION_ROLE system role.

The SYS_AUTH_DBA_ROLE compatibility role usually has more authority than is needed to synchronize. To set up a more secure synchronization environment, use one of the following approaches:

- Revoke SYS_AUTH_DBA_ROLE from the SYS_RUN_REPLICATION_ROLE system role and grant the following system privileges:
    - INSERT ANY TABLE
    - UPDATE ANY TABLE
    - DELETE ANY TABLE
    - ALTER ANY TABLE
    - EXECUTE ANY PROCEDURE

    - Any system privileges required by statements contained in hooks.
    - Any system privileges required by statements contained in stored procedures used to define scripted uploads.

  The advantages of this approach are simplicity and the fact these system-level privileges can only be used when the user is connected through an authenticated tool (like dbmlsync or SQL REMOTE). The disadvantage of this approach is that the SYS_RUN_REPLICATION_ROLE system role is granted more privileges than are strictly needed for synchronization. It is given INSERT, UPDATE, DELETE and ALTER privileges on all tables and EXECUTE on all procedures when it only needs these privileges on a few tables and procedures.
- Revoke SYS_AUTH_DBA_ROLE from the SYS_RUN_REPLICATION_ROLE system role, create a user-extended role that has the SYS_RUN_REPLICATION_ROLE system role, and grant the user-extended role to any user you want to allow to synchronize the database. Grant the following object-level privileges to the user-extended role:
    - INSERT, UPDATE, DELETE and ALTER on all tables that are to be synchronized.
    - EXECUTE on all hook procedures and on stored procedures used to define scripted uploads.
    - SELECT, INSERT, UPDATE and DELETE on the tables dbo.synchronize_results and dbo.synchronize_parameters. This is only required if the SQL SYNCHRONIZE statement is being used.
    - Any privileges required by statements contained in hooks.
    - Any privileges required by statements contained in stored procedures used to define scripted uploads.

  The advantage of this approach is that you have very fine control over the privileges granted to the user. However, the privileges granted are available to the users regardless of how they are logged in. Users are not just limited to connections made by dbmlsync and SQL Remote.

Example

The following examples shows how to revoke the SYS_AUTH_DBA_ROLE compatibility role from the SYS_RUN_REPLICATION_ROLE system role and grant the necessary privileges for synchronization.

```
REVOKE ROLE SYS_AUTH_DBA_ROLE FROM SYS_RUN_REPLICATION_ROLE;
GRANT INSERT ANY TABLE TO SYS_RUN_REPLICATION_ROLE;
GRANT UPDATE ANY TABLE TO SYS_RUN_REPLICATION_ROLE;
GRANT DELETE ANY TABLE TO SYS_RUN_REPLICATION_ROLE;
GRANT ALTER ANY TABLE TO SYS_RUN_REPLICATION_ROLE;
GRANT EXECUTE ANY PROCEDURE TO SYS_RUN_REPLICATION_ROLE;
```

The following example shows you how to revoke the SYS_AUTH_DBA_ROLE compatibility role from the SYS_RUN_REPLICATION_ROLE system role and create a user-extended role called **`SYNC_USER`** with sufficient privileges to synchronize the tables **`T1`** and **`T2`** and execute the sp_hook_dbmlsync_begin hook. It then grants the SYNC_USER role to the user **`user1`**.

```
REVOKE ROLE SYS_AUTH_DBA_ROLE FROM SYS_RUN_REPLICATION_ROLE;
// Create user-extended role SYNC_USER
CREATE USER SYNC_USER IDENTIFIED BY 'sql';
GRANT ROLE SYS_RUN_REPLICATION_ROLE TO SYNC_USER;
CREATE ROLE FOR USER SYNC_USER;
// Grant privileges on table T1 to SYNC_USER
GRANT INSERT ON T1 TO SYNC_USER;
GRANT UPDATE ON T1 TO SYNC_USER;
GRANT DELETE ON T1 TO SYNC_USER;
GRANT ALTER ON T1 TO SYNC_USER;
// Grant privileges on table T2 to SYNC_USER
```

```
GRANT INSERT ON T2 TO SYNC_USER;
GRANT UPDATE ON T2 TO SYNC_USER;
GRANT DELETE ON T2 TO SYNC_USER;
GRANT ALTER ON T2 TO SYNC_USER;
// Grant privileges on any hooks to SYNC_USER
GRANT EXECUTE ON dba.sp_hook_dbmlsync_begin TO SYNC_USER;
// Grant privileges on the synchronize_results and synchronize_parameters tables
// to SYNC_USER so that it can use the SYNCHRONIZE statement
GRANT SELECT ON dbo.synchronize_results TO SYNC_USER;
GRANT INSERT ON dbo.synchronize_results TO SYNC_USER;
GRANT UPDATE ON dbo.synchronize_results TO SYNC_USER;
GRANT DELETE ON dbo.synchronize_results TO SYNC_USER;
GRANT SELECT ON dbo.synchronize_parameters TO SYNC_USER;
GRANT INSERT ON dbo.synchronize_parameters TO SYNC_USER;
GRANT UPDATE ON dbo.synchronize_parameters TO SYNC_USER;
GRANT DELETE ON dbo.synchronize_parameters TO SYNC_USER;
// Grant SYNC_USER to user1 so that user1 can synchronize the database
GRANT ROLE SYNC_USER to user1;
```

Related Information

System Privileges
User-extended Roles
User Security (Roles and Privileges)
GRANT ROLE Statement

1.6.1.5.2 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 or the sp_hook_dbmlsync_set_extended_options hook procedure.

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"
```

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'
```

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

In this section:

Related Information

MobiLink SQL Anywhere Client Extended Options [page 160]
sp_hook_dbmlsync_set_extended_options [page 285]

1.6.1.5.2.1 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.

Related Information

CommunicationType (ctp) Extended Option [page 167]
CommunicationAddress (adr) Extended Option [page 166]

1.6.1.5.3 Transaction Log Files

Usually, dbmlsync determines what to upload by using the SQL Anywhere transaction log. SQL Anywhere databases maintain transaction logs by default. You can determine where the transaction log is located, or whether to have one, when you create the database or subsequently using the dblog utility.

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

To prepare the upload, the dbmlsync utility requires access to all transaction logs written since the last successful synchronization of all subscriptions for the MobiLink user who is synchronizing. However, SQL Anywhere transaction log files are typically truncated and renamed as part of regular database maintenance. In such a case, old transaction 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.

Example

Suppose that the old transaction 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.

Related Information

Database Backup and Recovery
Progress Offsets [page 84]
The Transaction Log
Scripted Upload [page 308]
Initialization Utility (dbinit)

1.6.1.5.4 Concurrency During Synchronization

To ensure the integrity of synchronizations, dbmlsync must ensure that no changes downloaded from the server modify rows in the remote database that have been changed since the last upload was sent.

By default, it usually does this without locking any tables so the impact on other concurrent users of the database is minimized. Tables are locked IN SHARE MODE when synchronizing a publication that uses scripted upload or when the sp_hook_dbmlsync_schema_upgrade hook is defined.

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

If a conflict is detected, the download phase is canceled and the download operations rolled back to avoid overwriting the new change. The 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.

-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. 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.

LockTables option

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

Related Information

ConflictRetries (cr) Extended Option [page 168]
-d dbmlsync Option [page 129]
LockTables (lt) Extended Option [page 179]

1.6.1.5.5 Synchronization Initiation 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 three ways to do this:

- Dbmlsync API
- SQL SYNCHRONIZE statement
- 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 17 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 Microsoft Windows and UNIX/Linux. The Dbmlsync API and the SQL SYNCHRONIZE statement are both easier to use than the DBTools interface and you are strongly encouraged to consider using them first.

Related Information

Dbmlsync API [page 113]
DBTools Interface for dbmlsync [page 303]
SYNCHRONIZE Statement [MobiLink]

1.6.1.6 Synchronization Schedules

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 continues running until stopped by the user.
- 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.

This method is not available when the Dbmlsync API or the SQL SYNCHRONIZE statement is used.

Hovering

When hovering, dbmlsync scans the database transaction log and builds its upload during the delay between synchronizations. This allows synchronization to proceed more quickly when it is triggered because some of the work is already done.

When hovering, dbmlsync scans to the end of the transaction log then polls the log periodically for new transactions. You can control the interval between polls using the PollingPeriod extended option of the -pp option.

When you are hovering on two or more subscriptions at the same time, you can use the HoverRescanThreshold extended option or the sp_hook_dbmlsync_log_rescan event hook to limit memory usage by recovering otherwise lost memory.

Hovering can be disabled using the DisablePolling extended option or the -p option.

In this section:

Related Information

PollingPeriod (pp) Extended Option [page 185]
HoverRescanThreshold (hrt) Extended Option [page 176]
-p dbmlsync Option [page 142]
DisablePolling (p) Extended Option [page 170]
sp_hook_dbmlsync_log_rescan [page 268]

1.6.1.6.1 Schedule Setup 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.

This method is not available when the Dbmlsync API or the SQL SYNCHRONIZE statement is used.

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.

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.

Related Information

MobiLink SQL Anywhere Client Extended Options [page 160]
Schedule (sch) Extended Option [page 186]
-eu dbmlsync Option [page 136]
-is dbmlsync Option [page 136]

1.6.1.6.2 Synchronization Initiation 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.

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.

This method is not available when the Dbmlsync API or the SQL SYNCHRONIZE statement is used.

Related Information

sp_hook_dbmlsync_end [page 265]
sp_hook_dbmlsync_delay [page 250]

1.6.1.7 dbmlsync Synchronization Customization

You can use event hooks, dbmlsync programming interfaces and scripted upload to customize dbmlsync synchronizations.

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.

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.

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.

Scripted upload

You can also override the use of the client transaction log and define your own upload stream.

Related Information

Event Hooks for SQL Anywhere Clients [page 232]
Dbmlsync API [page 113]
DBTools Interface for dbmlsync [page 303]
Scripted Upload [page 308]

1.6.1.8 Dbmlsync API

The Dbmlsync API provides a programming interface that allows MobiLink client applications written in C++ or .NET to launch synchronizations and receive feedback about the progress of the synchronizations they request. The API is intended to integrate synchronization seamlessly into your applications.

This programming interface enables you to access a lot more information about synchronization results and it also enables you to queue synchronizations, making them easier to manage.

⚠ Caution

The Dbmlsync API is not thread-safe. All calls to a single instance of the DbmlsyncClient class must be made on the same thread. Calling functions for a single instance of DbmlsyncClient from different threads can result in unexpected errors and unreliable results.

Related Information

Dbmlsync .NET API Reference [page 302]
Dbmlsync C++ API Reference

1.6.1.9 SQL Anywhere Client Logs

When you create MobiLink applications with SQL Anywhere remote databases, there are two types of client log files: the dbmlsync message log and the SQL Anywhere transaction log.

dbmlsync message log

By default, dbmlsync messages are sent to the dbmlsync messages 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 message log file named `dbmlsync.dbs`.

```
dbmlsync -o dbmlsync.dbs ...
```

Logging dbmlsync activity is particularly useful during the development process and when troubleshooting.

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

- Use the -o option to specify a message log file and append output to it.
- Use the -ot option to specify a message 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 message log file is renamed and a new file is started with the original name.

When no message log file is specified, all output is displayed in the dbmlsync messages window. When a message log file is specified, less output is sent to the dbmlsync messages window.

You can control what information is logged to the message log file and displayed in the dbmlsync messages window using the -v option. Verbose output is not recommended for normal operation in a production environment because it can slow performance.

SQL Anywhere transaction log

Usually, dbmlsync determines what to upload by using the SQL Anywhere transaction log. SQL Anywhere databases maintain transaction logs by default.

Related Information

Transaction Log Files [page 107]
-o dbmlsync Option [page 141]
-ot dbmlsync Option [page 142]
-os dbmlsync Option [page 141]
-v dbmlsync Option [page 157]

1.6.1.10 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 9.0.0, 9.0.1, and 9.0.2.

1.6.2 MobiLink SQL Anywhere Client Utility (dbmlsync) Syntax

Use the dbmlsync utility to synchronize SQL Anywhere remote databases with a consolidated database. To run dbmlsync you must have the SYS_RUN_REPLICATION_ROLE system privilege.

Syntax

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

Option	Description
@ `data`	Read in options from the specified environment variable or configuration file.
-a	Do not prompt for input again on error.
-ap `parameter,...`	Specify authentication parameters.
-ba `filename`	Apply a download file.
-bc `filename`	Create a download file.
-be `string`	When creating a download file, add a string.
-bg	When creating a download file, make it suitable for new remotes.
-bk	Enables background synchronization.
-bkr `number`	Specifies the number of times dbmlsync will retry synchronization after a background synchronization is interrupted.
-c `connection-string`	Supply database connection parameters in the form `parm1=value1`; `parm2=value2`,... that are used to connect to the remote database. If you do not supply this option, a window appears and you must supply connection information.
-ci `size`	Sets the initial size of the dbmlsync cache.

Option	Description
-cl `size`	Set the minimum size threshold for the dbmlsync cache file.
-cm `size`	Set the maximum size limit for the dbmlsync cache file.
-d	Drop any other connections to the database whose locks conflict with the articles to be synchronized.
-dc	Continue a previously failed download.
-dl	Display log messages on the dbmlsync messages window.
-do	Disables scanning of offline transaction logs. This utility option cannot be used with the -x utility option.
-drs `bytes`	For restartable downloads, specify the maximum amount of data that may need to be re-sent after a communications failure.
-ds	Perform a download-only synchronization.
-e "`keyword=value`"...	Specify extended options.
-eh	Ignore errors that occur in hook functions.
-ek `key`	Specify the remote database encryption key.
-ep "`keyword=value`"	Prompt for the remote database encryption key.
-eu	Specify extended options for upload defined by most recent -n option.
-is	Ignore schedule.
-k	Close window on completion.
-kpd	In the event of a download failure, saves the information required to attempt restarting the download.
-l	List available extended options.
-mn `password`	Specify new MobiLink password.
-mp `password`	Specify MobiLink password.
-n "`name, ...`"	Specify synchronization publication name(s).
-o `filename`	Log output messages to this file.
-os `size`	Specify a maximum size for the message log file, at which point the log is renamed.
-ot `logfile`	Delete the contents of the message log file and then log output messages to it.
-p	Disable logscan polling.
-pc[+ | -]	Maintain an open connection to the MobiLink server between synchronizations.
-pi	Test that you can connect to MobiLink.
-po `port`	Specifies the port on which dbmlsync listens.

Option	Description
-pp number	Set logscan polling period.
-q	Run in minimized window.
-qc	Shut down dbmlsync when synchronization is finished.
-qi	Starts dbmlsync in quiet mode with the window completely hidden.
-r[*a* | *b*]	Use client progress values for upload retry.
-s name	Specify synchronization subscription name(s).
-sc	Reload schema information before each synchronization.
-sm	Causes dbmlsync to start in server mode.
-sp sync-profile	Add options from the synchronization profile to the synchronization options specified on the command line.
-ts session-name(session-option=option-value[;...])	Sets up a MobiLink client tracing session.
-tu	Perform transactional upload.
-u ml_username	Specify the MobiLink user to synchronize.
-ud	For UNIX and Linux only. Run dbmlsync as a daemon.
-ui	For Linux with X window, starts dbmlsync in shell mode if a usable display isn't available.
-uo	Perform upload-only synchronization.
-urc row-estimate	Specify an estimate of the number of rows to upload.
-ux	For Solaris and Linux, open the dbmlsync messages window.
-v[levels]	Verbose operation.
-wc classname	Specify a window class name.
-x [size]	Rename and restart the transaction log. Use the optional size parameter with the -x option to control the size of the transaction log.
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 from the publication, synchronization user, synchronization subscription, or the dbmlsync command line.

Transaction log file

The `transaction-logs-directory` is the directory that contains the transaction log for the SQL Anywhere remote database. There is an active transaction log and zero or more transaction log archive files, all of which may be required by dbmlsync to determine what to upload. Specify this parameter if the transaction log archive files are in a different directory than the active transaction log.

dbmlsync event hooks

There are also dbmlsync client stored procedures that can help you customize the synchronization process.

Using dbmlsync

For more information about using dbmlsync, see the documentation for synchronization initiation.

In this section:

Drops conflicting locks to the remote database.

-dc dbmlsync Option [page 130]
: Restart a previously failed download.

-dl dbmlsync Option [page 131]
: Displays messages in the dbmlsync messages window or command prompt, and the message log file.

-do dbmlsync Option [page 131]
: Disables scanning of offline transaction logs.

-drs dbmlsync Option [page 132]
: For restartable downloads, specifies the maximum number of bytes that may need to be re-sent after a communications failure.

-ds dbmlsync Option [page 133]
: Performs a download-only synchronization.

-e dbmlsync Option [page 134]
: Specifies extended options.

-eh dbmlsync Option [page 135]
: Ignores errors that occur in hook functions.

-ek dbmlsync Option [page 135]
: Allows you to specify the encryption key for strongly encrypted remote databases directly on the command line.

-ep dbmlsync Option [page 135]
: Prompt for the encryption key for the remote database.

-eu dbmlsync Option [page 136]
: Specifies extended upload options.

-is dbmlsync Option [page 136]
: Ignores the Schedule extended option.

-k dbmlsync Option (Deprecated) [page 137]
: Shuts down dbmlsync when synchronization is finished. Dbmlsync does not shut down if an error occurs during the synchronization unless the -c or -ot option is also specified.

-kpd dbmlsync Option [page 137]
: Saves the information required to attempt restarting the failed download.

-l dbmlsync Option [page 138]
: Lists available extended options.

-mn dbmlsync Option [page 138]
: Supplies a new password for the MobiLink user being synchronized.

-mp dbmlsync Option [page 139]
: Supplies the password of the MobiLink user being synchronized.

-n dbmlsync Option (Deprecated) [page 140]
: Specifies the publication(s) to synchronize.

-o dbmlsync Option [page 141]
: Specifies the name of the dbmlsync message log file.

-os dbmlsync Option [page 141]

Specifies a maximum size for the dbmlsync message log file, at which point the message log file is renamed.

-ot dbmlsync Option [page 142]
: Deletes the contents of the specified file and then logs output messages to it.

-p dbmlsync Option [page 142]
: Disables logscan polling.

-pc dbmlsync Option [page 143]
: Maintain a persistent connection to the MobiLink server between synchronizations.

-pi dbmlsync Option [page 144]
: Pings a MobiLink server.

-po dbmlsync Option [page 145]
: When dbmlsync is in server mode, this option specifies the port on which dbmlsync listens for connections from clients.

-pp dbmlsync Option [page 145]
: Specifies the frequency of log scans.

-q dbmlsync Option [page 146]
: Starts the MobiLink synchronization client in a minimized window.

-qc dbmlsync Option [page 146]
: Shuts down dbmlsync when synchronization is finished.

-qi dbmlsync Option [page 147]
: Controls whether the dbmlsync system tray icon and messages window appear.

-r dbmlsync Option [page 147]
: Specifies that the remote offset should be used when there is disagreement between the offsets in the remote and consolidated databases.

-s dbmlsync Option [page 148]
: Specifies the subscription(s) to be synchronized.

-sc dbmlsync Option [page 149]
: Specifies that dbmlsync should reload schema information before each synchronization.

-sm dbmlsync Option [page 149]
: Causes dbmlsync to start in server mode.

-sp dbmlsync Option [page 150]
: When -sp is used, the options in the specified synchronization profile are added to those specified on the command line for the synchronization.

-ts dbmlsync Option [page 151]
: Sets up a MobiLink client tracing session.

-tu dbmlsync Option [page 152]
: Specifies that each transaction on the remote database should be uploaded as a separate transaction within one synchronization.

-u dbmlsync Option (Deprecated) [page 153]
: Specifies the MobiLink user name.

-ud dbmlsync Option [page 154]
: For UNIX and Linux platforms only, instructs dbmlsync to run as a daemon.

-ui dbmlsync Option [page 155]
  For Linux with X Windows server support, starts dbmlsync in shell mode if a usable display is not available.

-uo dbmlsync Option [page 155]
  Specifies that synchronization only includes an upload.

-urc dbmlsync Option [page 156]
  Specifies an estimate of the number of rows to be uploaded in a synchronization.

-ux dbmlsync Option [page 157]
  On Linux, opens a dbmlsync messages window where messages are displayed.

-v dbmlsync Option [page 157]
  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.

-wc dbmlsync Option [page 159]
  Specifies a window class name.

-x dbmlsync Option [page 159]
  Renames and restarts the transaction log.

Related Information

MobiLink SQL Anywhere Client Extended Options [page 160]
Initiation of Synchronization [page 102]
Transaction Log Files [page 107]
Event Hooks for SQL Anywhere Clients [page 232]
Initiation of Synchronization [page 102]
Event Hooks for SQL Anywhere Clients [page 232]
Dbmlsync API [page 113]
DBTools Interface for dbmlsync [page 303]
GRANT ROLE Statement

1.6.2.1 @data dbmlsync Option

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

Syntax

```
dbmlsync @data ...
```

Remarks

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

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

Related Information

Configuration Files
File Hiding Utility (dbfhide)

1.6.2.2 -a dbmlsync Option

Certain types of errors (such as an incorrect MobiLink password) normally cause dbmlsync to display a window prompting the user to correct the values. The -a option prevents dbmlsync from prompting after these errors occur.

Syntax

```
dbmlsync -a ...
```

1.6.2.3 -ap dbmlsync Option

Supplies parameters passed to the authenticate_parameters script and to authentication parameters on the MobiLink server.

Syntax

```
dbmlsync -ap "parameters,..." ...
```

Remarks

Use when you use the authenticate_parameters connection script or authentication parameters on the server. For example:

```
dbmlsync -ap "parm1,parm2,parm3"
```

The parameters are sent to the MobiLink server and passed to the authenticate_parameters script or other events on the consolidated database.

Related Information

Authentication Parameters
authenticate_parameters Connection Event
AuthParms Synchronization Profile Option [page 208]

1.6.2.4 -ba dbmlsync Option

Applies a download file.

Syntax

```
dbmlsync -ba "filename" ...
```

Remarks

Specify the name of an existing download file to be applied to the remote database. You can optionally specify a path. If you do not specify a path, the default location is the directory where dbmlsync was started.

Related Information

MobiLink File-based Download
-bc dbmlsync Option [page 123]
-be dbmlsync Option [page 124]
-bg dbmlsync Option [page 125]
ApplyDnldFile Synchronization Profile Option [page 209]

1.6.2.5 -bc dbmlsync Option

Creates a download file.

Syntax

```
dbmlsync -bc "filename" ...
```

Remarks

Create a download file with the specified name. You should use the file extension `.df` for download files.

You can optionally specify a path. If you do not specify a path, the default location is the dbmlsync current working directory, which is the directory where dbmlsync was started.

Optionally, when creating a download file, you can use the -be option to specify a string that can be validated at the remote database, and the -bg option to create a download file for new remote databases.

Related Information

MobiLink File-based Download
-ba dbmlsync Option [page 123]
-be dbmlsync Option [page 124]
-bg dbmlsync Option [page 125]
CreateDnldFile Synchronization Profile Option [page 212]

1.6.2.6 -be dbmlsync Option

When creating a download file, this option specifies an extra string to be included in the file.

Syntax

```
dbmlsync -bc "filename" -be "string" ...
```

Remarks

The string can be used for authentication or other purposes. It is passed to the sp_hook_dbmlsync_validate_download_file stored procedure on the remote database when the download file is applied.

Related Information

MobiLink File-based Download
sp_hook_dbmlsync_validate_download_file [page 298]
-bc dbmlsync Option [page 123]
-ba dbmlsync Option [page 123]
DnldFileExtra Synchronization Profile Option [page 213]

1.6.2.7 -bg dbmlsync Option

When creating a download file, this option creates a file that can be used by remote databases that have not yet synchronized.

Syntax

```
dbmlsync -bc "filename" -bg ...
```

Remarks

The -bg option causes the download file to update the generation numbers on the remote database.

This option allows you to build a download file that can be applied to remote databases that have never synchronized. Otherwise, you must perform a synchronization before you apply a download file.

Download files built with the -bg option should be snapshot downloads. Timestamp-based downloads do not work with remote databases that have not synchronized because the last download timestamp on a new remote database 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 database.

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.

Related Information

MobiLink File-based Download
MobiLink Generation Numbers
Application of the Download File
-ba dbmlsync Option [page 123]
-bc dbmlsync Option [page 123]
UpdateGenNum Synchronization Profile Option [page 228]

1.6.2.8 -bk dbmlsync Option

Enables background synchronization.

Syntax

```
dbmlsync -bk "connection-string" ...
```

Remarks

During a background synchronization, the database server drops the dbmlsync connection to the remote database and rolls back any uncommitted dbmlsync operations, if another connection is waiting for access to any database resource that dbmlsync has locked. This allows the other connections to go forward without waiting for the synchronization to complete. Depending on the operations dbmlsync had outstanding when its connection is dropped, there may still be a significant delay for the waiting connection as the database rolls back the dbmlsync uncommitted changes.

When the dbmlsync connection is dropped, the synchronization in progress will fail and report errors.

Related Information

MobiLink Synchronization Profiles [page 203]
-bkr dbmlsync Option [page 126]
Background Synchronization Profile Option [page 210]

1.6.2.9 -bkr dbmlsync Option

Controls the behavior of dbmlsync after a background synchronization is interrupted.

Syntax

```
dbmlsync -bkr num...
```

Remarks

`num` is an integer greater than or equal to -1.

If `num` is -1 then dbmlsync retries an interrupted synchronization until it completes, successfully or unsuccessfully, without being interrupted. If `num` is 0 then dbmlsync does not retry the interrupted synchronization. If `num` is greater than 0 then dbmlsync retries the synchronization up to `num` times until it completes. After `num` attempts, if the synchronization has not completed, then it is run as a foreground synchronization so it will complete without interruption.

By default BackgroundRetry is 0. It is an error to set BackgroundRetry to a non-zero value when the Background option has not been set to TRUE.

The BackgroundRetry is ignored when the dbmlsync API or the SQL SYNCHRONIZE statement is used.

Related Information

-bk dbmlsync Option [page 125]
BackgroundRetry Synchronization Profile Option [page 210]

1.6.2.10 -c dbmlsync Option

Specifies connection parameters for the remote database.

Syntax

```
dbmlsync -c "connection-string" ...
```

Remarks

The connection string must give dbmlsync permission to connect to the SQL Anywhere remote database as a user that has the SYS_REPLICATION_ADMIN_ROLE system role or equivalent privileges.

Specify the connection string in the form `keyword` = `value`, with multiple parameters separated by semicolons. If any of the parameter names contain spaces, you need to enclose the connection string in double quotes.

If you do not specify -c, a dbmlsync Setup window appears. You can specify the remaining command line options in the connection window fields.

Related Information

Alphabetical List of Connection Parameters

1.6.2.11 -ci dbmlsync Option

Sets the initial size of the dbmlsync cache.

Syntax

```
dbmlsync -ci size [ K | M | P ]...
```

Remarks

The `size` is the initial cache size, in bytes, used by dbmlsync to store synchronization data. You can optionally use the suffix K or M to specify units of kilobytes or megabytes, respectively.

To specify the size as a percentage of the total physical memory in the system, specify a number between 0 and 100, followed by the letter p. For example, `-ci 30p` sets the initial cache size to 30% of the physical memory.

Related Information

-cm dbmlsync Option [page 129]
-cl dbmlsync Option [page 128]

1.6.2.12 -cl dbmlsync Option

Set the minimum size to which the dbmlsync cache file is reduced.

Syntax

```
dbmlsync -cl size [ K | M | P ]...
```

Remarks

The `size` is the smallest size, in bytes, that the dbmlsync cache can be reduced to. You can optionally use the suffix K or M to specify units of kilobytes or megabytes, respectively.

To specify the size as a percentage of the total physical memory in the system, specify a number between 0 and 100, followed by the letter p. For example, `-cl 5p` ensures the cache size will not drop below 5% of the physical memory.

Related Information

-cm dbmlsync Option [page 129]
-ci dbmlsync Option [page 127]

1.6.2.13 -cm dbmlsync Option

Set the maximum size limit for the dbmlsync cache.

Syntax

```
dbmlsync -cm size [ K | M | P ]...
```

Remarks

The `size` is the largest size, in bytes, that the dbmlsync cache can grow to. You can optionally use the suffix K or M to specify units of kilobytes or megabytes, respectively.

To specify the size as a percentage of the total physical memory in the system, specify a number between 0 and 100, followed by the letter p. For example, `-cm 60p` limits the maximum size of the cache to 60% of physical memory.

Related Information

-ci dbmlsync Option [page 127]
-cl dbmlsync Option [page 128]

1.6.2.14 -d dbmlsync Option

Drops conflicting locks to the remote database.

Syntax

```
dbmlsync -d ...
```

Remarks

In cases where dbmlsync must obtain locks on the tables being synchronized, if another connection has a lock on one of these tables, the synchronization may fail or be delayed. Specifying this option forces SQL Anywhere to drop any other connections to the remote database that hold conflicting locks so that synchronization can proceed immediately.

Related Information

Concurrency During Synchronization [page 108]
KillConnections Synchronization Profile Option [page 219]

1.6.2.15 -dc dbmlsync Option

Restart a previously failed download.

Syntax

```
dbmlsync -dc ...
```

Remarks

This option can only be used if the -kpd dbmlsync option or the KeepPartialDownload synchronization profile option was specified on the synchronization that failed.

If dbmlsync does not receive the entire download from the server, dbmlsync does not apply any of the download data to the remote database. However, if the -kpd dbmlsync option was specified on the failed download, dbmlsync 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 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.

Related Information

Resumption of Failed Downloads
ContinueDownload (cd) Extended Option [page 169]
sp_hook_dbmlsync_end [page 265]
DownloadReadSize (drs) Extended Option [page 172]
ContinueDownload Synchronization Profile Option [page 211]

1.6.2.16 -dl dbmlsync 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 message 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 reduce the speed of synchronization.

1.6.2.17 -do dbmlsync 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 the -do option is used with dbmlsync, dbmlsync does not attempt to scan any offline transaction logs and should be able to synchronize from a database that is stored with all the other databases in a single directory.

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

This option cannot be used with -x option.

1.6.2.18 -drs dbmlsync Option

For restartable downloads, specifies the maximum number of bytes that may need to be re-sent 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 re-sent 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 re-sent 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.

Example

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

```
dbmlsync -drs 100
```

Related Information

Resumption of Failed Downloads
DownloadReadSize (drs) Extended Option [page 172]
ContinueDownload (cd) Extended Option [page 169]
sp_hook_dbmlsync_end [page 265]
-dc dbmlsync Option [page 130]

1.6.2.19 -ds dbmlsync Option

Performs a download-only synchronization.

Syntax

```
dbmlsync -ds ...
```

Remarks

When download-only synchronization occurs, dbmlsync does not upload any database changes. However, it does upload information about the schema and progress offset.

In addition, dbmlsync ensures that changes on the remote database 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 the documentation for required synchronization scripts.

Related Information

Upload-only and Download-only Synchronizations
Download-only Publications [page 90]
Scripts Required for Synchronization
DownloadOnly (ds) Extended Option [page 171]
DownloadOnly Synchronization Profile Option [page 214]

1.6.2.20 -e dbmlsync Option

Specifies extended options.

Syntax

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

Remarks

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

Use the dbmlsync -l option to get a list of all the extended options.

Options specified on the command line with the -e option apply to all synchronizations requested on the command line. For example, in the following command line the extended option drs=512 applies to the synchronization of both sub1 and sub2.

```
dbmlsync -e "drs=512" -s sub1 -s sub2
```

You can review extended options in the dbmlsync message log and the SYSSYNC system view.

To specify extended options for a single upload, use the -eu option.

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"...
```

Related Information

MobiLink SQL Anywhere Client Extended Options [page 160]
-l dbmlsync Option [page 138]
-eu dbmlsync Option [page 136]
SYSSYNC System View
sp_hook_dbmlsync_set_extended_options [page 285]
ExtOpt Synchronization Profile Option [page 216]

1.6.2.21 -eh dbmlsync Option

Ignores errors that occur in hook functions.

Syntax

```
dbmlsync -eh ...
```

Related Information

IgnoreHookErrors Synchronization Profile Option [page 217]

1.6.2.22 -ek dbmlsync Option

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

Syntax

```
dbmlsync -ek key ...
```

Remarks

If you have a strongly encrypted remote database, you must provide the encryption key to use the database or transaction log in any way, including offline transactions. For strongly encrypted databases, you must specify either -ek or -ep, but not both. The command fails if you do not specify a key for a strongly encrypted database.

1.6.2.23 -ep dbmlsync Option

Prompt for the encryption key for the remote database.

Syntax

```
dbmlsync -ep ...
```

Remarks

This option causes a window to appear, in which you enter the encryption key. It provides an extra measure of security by never allowing the encryption key to be seen in clear text. For strongly encrypted remote databases, you must specify either -ek or -ep, but not both. The command fails if you do not specify a key for a strongly encrypted database.

1.6.2.24 -eu dbmlsync Option

Specifies extended upload options.

Syntax

```
dbmlsync -s subscription-name -eu keyword=value;...
```

```
dbmlsync -n publication-name -eu keyword=value;...
```

Remarks

Extended options that are specified on the command line with the -eu option apply only to the synchronization specified by the -n option or the -s option they follow. For example, on the following command line, the extended option eh=on applies only to the synchronization of subscription sub2.

```
dbmlsync -s sub1 -s sub2 -eu eh=on
```

For an explanation of how extended options are processed when they are set in more than one place, and a complete list of extended options, see the documentation for MobiLink SQL Anywhere client extended options.

Related Information

MobiLink SQL Anywhere Client Extended Options [page 160]

1.6.2.25 -is dbmlsync Option

Ignores the Schedule extended option.

Syntax

```
dbmlsync -is ...
```

Remarks

Ignore extended options that schedule synchronization.

Related Information

Synchronization Schedules [page 110]
IgnoreScheduling Synchronization Profile Option [page 217]

1.6.2.26 -k dbmlsync Option (Deprecated)

Shuts down dbmlsync when synchronization is finished. Dbmlsync does not shut down if an error occurs during the synchronization unless the -c or -ot option is also specified.

This option is deprecated. Use -qc instead.

Syntax

```
dbmlsync -k  ...
```

Related Information

-qc dbmlsync Option [page 146]

1.6.2.27 -kpd dbmlsync Option

Saves the information required to attempt restarting the failed download.

Syntax

```
dbmlsync -kpd...
```

Remarks

If neither this option nor the KeepPartialDownload synchronization profile option is used, then dbmlsync does not save the portion of the download it received before the synchronization failed and the failed download cannot be restarted.

Restart a failed download by using one of the following methods:

- -dc dbmlsync option
- ContinueDownload extended option
- ContinueDownload synchronization profile option
- the restart parameter in the sp_hook_dbmlsync_end stored procedure

If neither this option nor the KeepPartialDownload synchronization profile option is used, then dbmlsync does not save the portion of the download that it received before the synchronization failed and the failed download cannot be restarted.

Related Information

-dc dbmlsync Option [page 130]
ContinueDownload (cd) Extended Option [page 169]
ContinueDownload Synchronization Profile Option [page 211]
sp_hook_dbmlsync_end [page 265]
KeepPartialDownload Synchronization Profile Option [page 218]

1.6.2.28 -l dbmlsync Option

Lists available extended options.

Syntax

```
dbmlsync -l ...
```

Related Information

MobiLink SQL Anywhere Client Extended Options [page 160]

1.6.2.29 -mn dbmlsync Option

Supplies a new password for the MobiLink user being synchronized.

Syntax

```
dbmlsync -mn password ...
```

Remarks

Changes the MobiLink user's password.

Related Information

MobiLink Users in a Synchronization System [page 9]
MobiLinkPwd (mp) Extended Option [page 181]
NewMobiLinkPwd (mn) Extended Option [page 182]
-mp dbmlsync Option [page 139]
NewMobiLinkPwd Synchronization Profile Option [page 222]

1.6.2.30 -mp dbmlsync Option

Supplies the password of the MobiLink user being synchronized.

Syntax

```
dbmlsync -mp password ...
```

Remarks

Supplies the password for MobiLink user authentication.

Related Information

MobiLink Users in a Synchronization System [page 9]
MobiLinkPwd (mp) Extended Option [page 181]
NewMobiLinkPwd (mn) Extended Option [page 182]
-mn dbmlsync Option [page 138]

1.6.2.31 -n dbmlsync Option (Deprecated)

Specifies the publication(s) to synchronize.

i Note

This option has been deprecated. Use the -s dbmlsync option instead.

To use the dbmlsync -s option you need to determine the subscription name for the subscription you want to synchronize. You can determine the subscription name using the following query:

```
SELECT subscription_name
FROM syssync JOIN sys.syspublication
WHERE site_name = <ml_user> AND publication_name = <pub_name>;
```

Replace <ml_user> with the MobiLink user you are synchronizing. This is the value specified by the -u option on the dbmlsync command line.

Replace <pub_name> with the name of the publication being synchronized. This is the value specified with the -n option on the dbmlsync command line.

Syntax

```
dbmlsync -n pubname ...
```

Remarks

You can supply more than one -n option to synchronize more than one synchronization publication.

There are two ways to use -n to synchronize multiple publications:

- Specify `-n pub1,pub2,pub3` to upload pub1, pub2, and pub3 in one upload followed by one download. In this case, if you have set extended options on the publications or subscriptions, only the options set on the first publication in the list and its subscription are used. Extended options set on subsequent publications and subscriptions are ignored.
- Specify `-n pub1 -n pub2 -n pub3` to synchronize pub1, pub2, and pub3 in three separate sequential synchronizations.
  When successive synchronizations occur very quickly, such as when you specify `-n pub1 -n pub2`, dbmlsync could start processing a synchronization when the server is still processing the previous synchronization. In this case, the second synchronization fails with an error indicating that concurrent synchronizations are not allowed. If you run into this situation, you can define an sp_hook_dbmlsync_delay stored procedure to create a delay before each synchronization. Usually a few seconds to a minute is a sufficient delay.

Related Information

sp_hook_dbmlsync_delay [page 250]

Publication Synchronization Profile Option (Deprecated) [page 223]
-s dbmlsync Option [page 148]
-u dbmlsync Option (Deprecated) [page 153]

1.6.2.32 -o dbmlsync Option

Specifies the name of the dbmlsync message log file.

Syntax

```
dbmlsync -o filename ...
```

Remarks

Append output to a dbmlsync message log file. Default is to send output to the screen.

Related Information

-os dbmlsync Option [page 141]
-ot dbmlsync Option [page 142]

1.6.2.33 -os dbmlsync Option

Specifies a maximum size for the dbmlsync message log file, at which point the message log file is renamed.

Syntax

```
dbmlsync -os size [ K | M | G ]...
```

Remarks

The `size` is the maximum file size for the dbmlsync message logs, specified in units of bytes. Use the suffix k, m or g to specify units of kilobytes, megabytes or gigabytes, respectively. By default, there is no size limit. The minimum size limit is 10K.

Before the dbmlsync utility logs output messages to a file, it checks the current file size. If the log message makes the file size exceed the specified size, the dbmlsync utility renames the output file to `yymmddxx`.dbs, where `yymmdd` represents the year, month, and day, and `xx` is a number that starts at 00 and continues incrementing.

This option allows you to manually delete old message log files and free up disk space.

Related Information

-o dbmlsync Option [page 141]
-ot dbmlsync Option [page 142]

1.6.2.34 -ot dbmlsync Option

Deletes the contents of the specified file and then logs output messages to it.

Syntax

```
dbmlsync -ot logfile ...
```

Remarks

The functionality is the same as the -o option except the contents of the message log file are deleted when dbmlsync starts up, before any messages are written to it.

Related Information

-o dbmlsync Option [page 141]
-os dbmlsync Option [page 141]

1.6.2.35 -p dbmlsync Option

Disables logscan polling.

Syntax

```
dbmlsync -p ...
```

Remarks

To build an upload, dbmlsync must scan the transaction log. Usually it does this at the beginning of synchronization. However, when synchronizations are scheduled or when the sp_hook_dbmlsync_delay hook is used, dbmlsync by default scans the log in the pause that occurs before synchronization. This behavior is more efficient because when synchronization begins the log is already at least partially scanned. This default behavior is called logscan polling.

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

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

Related Information

DisablePolling (p) Extended Option [page 170]
PollingPeriod (pp) Extended Option [page 185]
-pp dbmlsync Option [page 145]

1.6.2.36 -pc dbmlsync Option

Maintain a persistent connection to the MobiLink server between synchronizations.

Syntax

```
dbmlsync -pc [ + | - ] ...
```

Remarks

When -pc+ is used, dbmlsync connects to the MobiLink server as usual, but it then keeps that connection open for use during subsequent synchronizations. A persistent connection is closed when any of the following occur:

- An error occurs that causes a synchronization to fail.
- Liveness checking has timed out.
- 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.

The -pc+ option is most useful when the client synchronizes frequently and the cost of establishing a connection to the server is high.

When -pc- is used, the connection is closed at the end of each synchronization and reopened at the start of the next synchronization. By default, persistent connections are not maintained.

Related Information

timeout MobiLink Client Network Protocol Option [page 63]

1.6.2.37 -pi dbmlsync 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

Related Information

Ping Synchronization Profile Option [page 222]

1.6.2.38 -po dbmlsync Option

When dbmlsync is in server mode, this option specifies the port on which dbmlsync listens for connections from clients.

Syntax

```
dbmlsync -po port number ...
```

Remarks

This option can only be used with the -sm option.

Related Information

-sm dbmlsync Option [page 149]

1.6.2.39 -pp dbmlsync Option

Specifies the frequency of log scans.

Syntax

```
dbmlsync -pp number [ h | m | s ]...
```

Remarks

To build an upload, dbmlsync must scan the transaction log. Usually it does this at the beginning of synchronization. However, when synchronizations are scheduled or when the sp_hook_dbmlsync_delay hook is used, dbmlsync by default scans the log in the pause that occurs before synchronization. This behavior is more efficient because when synchronization begins the log is already at least partially scanned. This default behavior is called logscan polling.

This option specifies the interval between log scans. Use the suffix s, m, h, or d to specify seconds, minutes, hours or days, respectively. The default is *1* minute. If you do not specify a suffix, the default unit of time is minutes.

Related Information

PollingPeriod (pp) Extended Option [page 185]
DisablePolling (p) Extended Option [page 170]
-p dbmlsync Option [page 142]

1.6.2.40 -q dbmlsync Option

Starts the MobiLink synchronization client in a minimized window.

Syntax

```
dbmlsync -q ...
```

1.6.2.41 -qc dbmlsync Option

Shuts down dbmlsync when synchronization is finished.

Syntax

```
dbmlsync -qc ...
```

Remarks

When used, dbmlsync exits after synchronization is completed if the synchronization was successful or if a message log file was specified using the -o or -ot options.

Related Information

-o dbmlsync Option [page 141]
-ot dbmlsync Option [page 142]

1.6.2.42 -qi dbmlsync Option

Controls whether the dbmlsync system tray icon and messages window appear.

Syntax

```
dbmlsync -qi ...
```

Remarks

This option leaves no visual indication that dbmlsync is running, other than possible startup error windows. You can use the -o log files to diagnose errors.

When the -qi option is specified, dbmlsync exits after synchronization is complete.

Related Information

-o dbmlsync Option [page 141]
-ot dbmlsync Option [page 142]

1.6.2.43 -r dbmlsync Option

Specifies that the remote offset should be used when there is disagreement between the offsets in the remote and consolidated databases.

The -rb option indicates that the remote offset should be used if it is less than the consolidated offset (such as when the remote database has been restored from backup). The -r option is provided for backward compatibility and is identical to -rb. The -ra option indicates that the remote offset should be used if it is greater than the consolidated offset. This option is provided only for very rare circumstances and may cause data loss.

Syntax

```
dbmlsync { -r | -ra | -rb } ...
```

Remarks

-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 database is less than that obtained from the consolidated database. If you use -rb and the offset in the remote database 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 database 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.

Related Information

Progress Offsets [page 84]
RemoteProgressGreater Synchronization Profile Option [page 224]
RemoteProgressLess Synchronization Profile Option [page 225]

1.6.2.44 -s dbmlsync Option

Specifies the subscription(s) to be synchronized.

Syntax

```
dbmlsync -s subname ...
```

Remarks

This option replaces the -n dbmlsync option.

There are two ways to use -s to synchronize multiple subscriptions:

- Specify `-s sub1,sub2,sub3` to synchronize sub1, sub2, and sub3 in one upload followed by one download.
  In this case, if you have set extended options on the subscriptions, only the options set on the first subscription in the list are used. Extended options set on subsequent subscriptions ignored.

- Specify `-s sub1 -s sub2 -s sub3` to synchronize sub1, sub2, and sub3 in three separate sequential synchronizations, each with its own upload and download.
  When successive synchronizations occur very quickly, such as when you specify `-s sub1 -s sub2`, dbmlsync could start processing a synchronization when the server is still processing the previous synchronization. In this case, the second synchronization fails with an error indicating that concurrent synchronizations are not allowed. If you run into this situation, you can define an sp_hook_dbmlsync_delay stored procedure to create a delay before each synchronization. Usually a few seconds to a minute is a sufficient delay.

Related Information

sp_hook_dbmlsync_delay [page 250]
Subscription Synchronization Profile Option [page 226]

1.6.2.45 -sc dbmlsync Option

Specifies that dbmlsync should reload schema information before each synchronization.

Syntax

```
dbmlsync -sc ...
```

Remarks

Before 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 often 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.

1.6.2.46 -sm dbmlsync Option

Causes dbmlsync to start in server mode.

Syntax

```
dbmlsync -sm ...
```

Remarks

When in server mode, dbmlsync starts up and waits for connections from applications using the Dbmlsync API or the SQL SYNCHRONIZE statement.

This option should only be used when starting a dbmlsync server from the command line.

Normally dbmlsync is started directly using the Dbmlsync API or the SQL SYNCHRONIZE statement. This option **should not** be used when either of those methods is used.

Related Information

-po dbmlsync Option [page 145]

1.6.2.47 -sp dbmlsync Option

When -sp is used, the options in the specified synchronization profile are added to those specified on the command line for the synchronization.

Syntax

```
dbmlsync -sp sync profile ...
```

Remarks

If equivalent options are specified on the command line and in the synchronization profile, then the options on the command line override those specified in the profile.

Related Information

MobiLink Synchronization Profiles [page 203]

CREATE SYNCHRONIZATION PROFILE Statement [MobiLink]
ALTER SYNCHRONIZATION PROFILE Statement [MobiLink]
DROP SYNCHRONIZATION PROFILE Statement [MobiLink]

1.6.2.48 -ts dbmlsync Option

Sets up a MobiLink client tracing session.

Syntax

```
dbmlsync -ts session-name(session-option=option-value[;...])
```

The session name must be *logging*.

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

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

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

Remarks

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

You can retrieve data contained in the ETD file using either the Event Trace Data (ETD) File management utility (dbmanageetd) or the sp_read_etd system procedure.

Example

Following is an example of the -ts option:

```
-ts
logging(events=Info,warning,Error;targets=file(filename_prefix=mls_etd;max_size=1
0000000;num_files=10;flush_on_write=true))
```

Related Information

Event Tracing
Viewing the Contents of the Event Trace Data (ETD) Diagnostic Log File
Event Trace Data (ETD) File Management Utility (dbmanageetd)
sp_read_etd System Procedure

1.6.2.49 -tu dbmlsync Option

Specifies that each transaction on the remote database should be uploaded as a separate transaction within one synchronization.

Syntax

```
dbmlsync -tu ...
```

Remarks

When you use -tu, you create a **transactional upload**: dbmlsync uploads each transaction on the remote database as a distinct transaction. The MobiLink server applies and commits each transaction separately when it is received.

When you use -tu, the order of transactions on the remote database is always preserved on the consolidated database. However, the order of operations in a transaction may not be preserved, for two reasons:

- MobiLink always applies updates based on foreign key relationships. For example, when data is changed in foreign key and primary key tables, MobiLink inserts data into the primary key table before the foreign key table, but deletes data from the foreign key table before the primary key table. If your remote operations do not follow this order, the order of operations differ on the consolidated database.
- Operations within a transaction are coalesced so if you change the same row three times in one transaction, only the final form of the row is uploaded.

If a transactional upload is interrupted, the data that was not sent is sent in the next synchronization. Typically, only the transactions that were not successfully completed are sent at that time. Sometimes, such as when the upload failure occurs during the first synchronization of a subscription, dbmlsync resends all transactions.

When you do not use -tu, MobiLink coalesces all changes on the remote database into one transaction in the upload so 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.

Related Information

Self-referencing Tables
-tx mlsrv17 Option
TransactionalUpload Synchronization Profile Option [page 227]

1.6.2.50 -u dbmlsync Option (Deprecated)

Specifies the MobiLink user name.

i Note

This option has been deprecated. Use the -s dbmlsync option instead.

To use the dbmlsync -s option you need to determine the subscription name for the subscription you want to synchronize. You can determine the subscription name using the following query:

```
SELECT subscription_name
FROM syssync JOIN sys.syspublication
WHERE site_name = ml_user AND publication_name = pub_name;
```

Replace `ml_user` with the MobiLink user you are synchronizing. This is the value specified by the -u option on the dbmlsync command line.

Replace `pub_name` with the name of the publication being synchronized. This is the value specified with the -n option on the dbmlsync command line.

Syntax

```
dbmlsync -u ml_username ...
```

Remarks

You can specify only one user on the dbmlsync command line, where `ml_username` is the name used in the FOR clause of the CREATE SYNCHRONIZATION SUBSCRIPTION statement corresponding to the subscription to be processed.

This option should be used 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.

Related Information

MLUser Synchronization Profile Option (Deprecated) [page 221]
-n dbmlsync Option (Deprecated) [page 140]
-s dbmlsync Option [page 148]

1.6.2.51 -ud dbmlsync Option

For UNIX and Linux platforms only, instructs dbmlsync to run as a daemon.

Syntax

```
dbmlsync -ud ...
```

Remarks

If you run dbmlsync as a daemon, you should also supply either the -o or -ot option to log output information.

When you start dbmlsync as a daemon, its permissions are controlled by the current user's umask setting. Set the umask value before starting dbmlsync to ensure that dbmlsync has the appropriate permissions.

Related Information

-o dbmlsync Option [page 141]
-ot dbmlsync Option [page 142]

1.6.2.52 -ui dbmlsync Option

For Linux with X Windows server support, starts dbmlsync in shell mode if a usable display is not available.

Syntax

```
dbmlsync  -ui  ...
```

Remarks

When this option is used, dbmlsync tries to start with X Windows. If this fails, it starts in shell mode.

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

1.6.2.53 -uo dbmlsync Option

Specifies that synchronization only includes an upload.

Syntax

```
dbmlsync  -uo...
```

Remarks

During upload-only synchronization, dbmlsync prepares and sends an upload to MobiLink exactly as it would in a normal full synchronization. However, instead of sending a download back down, MobiLink sends only an acknowledgement indicating if the upload was successfully committed.

For a list of the scripts that must be defined for upload-only synchronization, see the documentation about scripts required during synchronization.

Related Information

Upload-only and Download-only Synchronizations
Scripts Required for Synchronization
DownloadOnly (ds) Extended Option [page 171]
UploadOnly (uo) Extended Option [page 194]
UploadOnly Synchronization Profile Option [page 229]

1.6.2.54 -urc dbmlsync Option

Specifies an estimate of the number of rows to be uploaded in a synchronization.

Syntax

```
dbmlsync -urc row-estimate ...
```

Remarks

To improve performance, you can specify an estimate of the number of rows to upload in a synchronization. This setting is especially useful when you are uploading a large number of rows. A higher estimate results in faster uploads but more memory usage.

Synchronization proceeds correctly regardless of the specified estimate.

Related Information

For Large Uploads, Estimate the Number of Rows
UploadRowCnt Synchronization Profile Option [page 229]

1.6.2.55 -ux dbmlsync Option

On Linux, opens a dbmlsync messages window where messages are displayed.

Syntax

```
dbmlsync -ux...
```

Remarks

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

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

On Windows, the dbmlsync messages window appears automatically.

Related Information

-q dbmlsync Option [page 146]

1.6.2.56 -v dbmlsync Option

Allows you to specify what information is logged to the message log file and displayed in the synchronization window. A high level of verbosity may affect performance and should normally be used in the development phase only.

Syntax

```
dbmlsync -v [ levels ] ...
```

Remarks

The -v options affect the message log file and synchronization window. You only have a message log if you specify -o or -ot on the dbmlsync command line.

If you specify -v alone, a small amount of information is logged but more information than if -v is omitted.

The values of `levels` are as follows. You can use one or more of these options at once; for example, -vnrsu or -v+cp.

+

Turn on all logging options except for c and p.

c

Expose the connection string in the log.

p

Expose the MobiLink password in the log.

n

Log the number of rows that were uploaded and downloaded.

o

Log information about the command line options and extended options that you have specified.

r

Log the values of rows that were uploaded and downloaded.

s

Log messages related to hook scripts.

u

Log information about the upload.

There are extended options that have similar functionality to the -v options. If you specify both -v and the extended options and there are conflicts, the -v option overrides the extended option. If there is no conflict, the verbosity logging options are additive and all options that you specify are used. If you use the -v command line option, the verbosity options take effect immediately. If you use the extended option, the verbosity options do not take effect until the first synchronization begins. After the first synchronization, the behavior should be the same regardless of how the option was specified.

Related Information

Verbose (v) Extended Option [page 195]
VerboseHooks (vs) Extended Option [page 196]
VerboseMin (vm) Extended Option [page 197]
VerboseOptions (vo) Extended Option [page 198]
VerboseRowCounts (vn) Extended Option [page 199]
VerboseRowValues (vr) Extended Option [page 200]
-o dbmlsync Option [page 141]
-ot dbmlsync Option [page 142]
Verbosity Synchronization Profile Option [page 230]

1.6.2.57 -wc dbmlsync Option

Specifies a window class name.

Syntax

```
dbmlsync -wc class-name ...
```

Remarks

This option specifies a window class name that can be used to wake up dbmlsync whenever it is in hover mode, such as when scheduling is enabled or when you are using server-initiated synchronization.

This option applies only to Windows.

Example

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

Related Information

Synchronization Schedules [page 110]
Schedule (sch) Extended Option [page 186]

1.6.2.58 -x dbmlsync Option

Renames and restarts the transaction log.

Syntax

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

Remarks

You should always specify a size value with the -x option. Specifically, it is recommended that you specify *-x 0*. Specifying the size avoids potential ambiguity and unintended behavior. When the -x option is specified immediately before the offline-log directory, the size must be specified to avoid errors or unintended behavior.

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

If backups are not routinely performed at the remote database, the transaction log continues to grow. As an alternative to using the -x option to control transaction log size, you can use a SQL Anywhere event handler to control the size of the transaction log.

Related Information

Task Automation Using Schedules and Events
delete_old_logs Option [MobiLink][SQL Remote]
CREATE EVENT Statement

1.6.3 MobiLink SQL Anywhere Client 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 SQL 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)

Summary of dbmlsync extended options

Following is a list of the dbmlsync extended options.

Option	Description
BufferDownload={*ON* | *OFF*}; ...	Specifies whether the entire download from the MobiLink server should be read into the cache before applying it to the remote database.
CommunicationAddress= `protocol-option`; ...	Specifies network protocol options for connecting to the MobiLink server.
CommunicationType= `network-protocol`; ...	Specifies the type of network protocol to use for connecting to the MobiLink server.
ConflictRetries= `number`; ...	Specifies the number of retries if the download fails because of conflicts.
ContinueDownload={ *ON* | *OFF* }; ...	Restarts a previously failed download.
DisablePolling={*ON* | *OFF*}; ...	Disables automatic logscan polling.
DownloadOnly={ *ON* | *OFF* }; ...	Specifies that synchronization should be download-only.
DownloadReadSize= `number`[*K*]; ...	For restartable downloads, specifies the maximum amount of data that may need to be re-sent after a communications failure.
ErrorLogSendLimit= `number`[*K* | *M*]; ...	Specifies how much of the remote message log file dbmlsync should send to the server when synchronization error occurs.
FireTriggers={ *ON* | *OFF* }; ...	Specifies that triggers should be fired on the remote database when the download is applied.
HoverRescanThreshold= `number`[*K* | *M*]; ...	When you are using scheduling, this limits the amount of discarded memory that is allowed to accumulate before a rescan is performed.
IgnoreHookErrors={ *ON* | *OFF* }; ...	Specifies that errors that occur in hook functions should be ignored.
IgnoreScheduling={ *ON* | *OFF* }; ...	Specifies that the Schedule extended option should be ignored.
Increment= `number`[*K* | *M*]; ...	Enables incremental uploads and controls the size of upload increments.
LockTables={ *ON* | *OFF* | *SHARE* | *EXCLUSIVE* }; ...	Specifies that tables in the publications being synchronized should be locked before synchronizing.
MirrorLogDirectory= `dir`; ...	Specifies the location of old transaction log mirror files so that they can be deleted.
MobiLinkPwd= `password`; ...	Specifies the MobiLink password.
NewMobiLinkPwd= `new-password`; ...	Specifies a new MobiLink password.
NoSyncOnStartup={ *on* | *off* }; ...	Prevents dbmlsync from synchronizing on startup when a scheduling option would otherwise cause that to happen.
OfflineDirectory= `path`; ...	Specifies the path containing offline transaction logs.
PollingPeriod= `number`[*S* | *M* | *H* | *D*]; ...	Specifies the logscan polling period.
Schedule= `schedule`; ...	Specifies a schedule for synchronization.
ScriptVersion= `version-name`; ...	Specifies a script version.

Option	Description
SendDownloadAck={ ON | OFF }; ...	Specifies that a download acknowledgement should be sent from the client to the server.
SendTriggers={ ON | OFF }; ...	Specifies that trigger actions should be sent on upload.
TableOrder= `tables`; ...	Specifies the order of tables in the upload.
TableOrderChecking={ OFF | ON }; ...	Lets you disable referential integrity checking on the table order specified by the TableOrder extended option.
UploadOnly={ ON | OFF }; ...	Specifies that synchronization should only include an up-load.
Verbose={ ON | OFF }; ...	Specifies full verbosity.
VerboseHooks={ ON | OFF }; ...	Specifies that messages related to hook scripts should be logged.
VerboseMin={ ON | OFF }; ...	Specifies that a small amount of information should be log-ged.
VerboseOptions={ ON | OFF }; ...	Specifies that information should be logged about the com-mand line options (including extended options) that you have specified.
VerboseRowCounts={ ON | OFF }; ...	Specifies that the number of rows that are uploaded and downloaded should be logged.
VerboseRowValues={ ON | OFF }; ...	Specifies that the values of rows that are uploaded and downloaded should be logged.
VerboseUpload={ ON | OFF }; ...	Specifies that information about the upload steam should be logged.

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
```

In this section:

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.

BufferDownload (bd) Extended Option [page 166]
: Specifies whether the entire download from the MobiLink server should be read into the cache before applying it to the remote database.

CommunicationAddress (adr) Extended Option [page 166]
: Specifies network protocol options for connecting to the MobiLink server.

CommunicationType (ctp) Extended Option [page 167]
: Specifies the type of network protocol to use for connecting to the MobiLink server.

ConflictRetries (cr) Extended Option [page 168]
: Specifies the number of retries if the download fails because of conflicts.

ContinueDownload (cd) Extended Option [page 169]
: Restarts a previously failed download.

DisablePolling (p) Extended Option [page 170]
: Disables automatic logscan polling.

DownloadOnly (ds) Extended Option [page 171]
: Specifies that synchronization should be download-only.

DownloadReadSize (drs) Extended Option [page 172]
: For restartable downloads, specifies the maximum amount of data that may need to be re-sent after a communications failure.

ErrorLogSendLimit (el) Extended Option [page 174]
: Specifies how much of the remote message log file dbmlsync should send to the server when synchronization error occurs.

FireTriggers (ft) Extended Option [page 175]
: Specifies that triggers should be fired on the remote database when the download is applied.

HoverRescanThreshold (hrt) Extended Option [page 176]
: When you are using scheduling, this limits the amount of discarded memory that is allowed to accumulate before a rescan is performed.

IgnoreHookErrors (eh) Extended Option [page 177]
: Specifies that errors that occur in hook functions should be ignored.

IgnoreScheduling (isc) Extended Option [page 177]
: Specifies that the Schedule extended option should be ignored.

Increment (inc) Extended Option [page 178]
: Enables incremental uploads and controls the size of upload increments.

LockTables (lt) Extended Option [page 179]
: Specifies that tables in the publications being synchronized should be locked before synchronizing.

MirrorLogDirectory (mld) Extended Option [page 180]
: Specifies the location of old transaction log mirror files so that they can be deleted.

MobiLinkPwd (mp) Extended Option [page 181]
: Specifies the MobiLink password.

NewMobiLinkPwd (mn) Extended Option [page 182]

Specifies a new MobiLink password.

NoSyncOnStartup (nss) Extended Option [page 183]
: Prevents dbmlsync from synchronizing on startup when a scheduling option would otherwise cause that to happen.

OfflineDirectory (dir) Extended Option [page 184]
: Specifies the path containing offline transaction logs.

PollingPeriod (pp) Extended Option [page 185]
: Specifies the logscan polling period.

Schedule (sch) Extended Option [page 186]
: Specifies a schedule for synchronization.

ScriptVersion (sv) Extended Option [page 188]
: Specifies a script version.

SendDownloadAck (sa) Extended Option [page 189]
: Specifies that a download acknowledgement should be sent from the client to the server.

SendTriggers (st) Extended Option [page 190]
: Specifies that trigger actions should be sent on upload.

TableOrder (tor) Extended Option [page 192]
: Specifies the order of tables in the upload.

TableOrderChecking (toc) Extended Option [page 193]
: Lets you disable referential integrity checking on the table order specified by the TableOrder extended option.

UploadOnly (uo) Extended Option [page 194]
: Specifies that synchronization should only include an upload.

Verbose (v) Extended Option [page 195]
: Specifies full verbosity.

VerboseHooks (vs) Extended Option [page 196]
: Specifies that messages related to hook scripts should be logged.

VerboseMin (vm) Extended Option [page 197]
: Specifies that a small amount of information should be logged.

VerboseOptions (vo) Extended Option [page 198]
: Specifies that information should be logged about the command line options (including extended options) that you have specified.

VerboseRowCounts (vn) Extended Option [page 199]
: Specifies that the number of rows that are uploaded and downloaded should be logged.

VerboseRowValues (vr) Extended Option [page 200]
: Specifies that the values of rows that are uploaded and downloaded should be logged.

VerboseUpload (vu) Extended Option [page 201]
: Specifies that information about the upload steam should be logged.

Related Information

dbmlsync Extended Options [page 106]

-e dbmlsync Option [page 134]

-eu dbmlsync Option [page 136]

CREATE SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink]
ALTER SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink]
CREATE SYNCHRONIZATION USER Statement [MobiLink]
ALTER SYNCHRONIZATION USER Statement [MobiLink]
CREATE PUBLICATION Statement [MobiLink] [SQL Remote]
ALTER PUBLICATION Statement [MobiLink] [SQL Remote]
SYSSYNC System View

sp_hook_dbmlsync_set_extended_options [page 285]

1.6.3.1 How dbmlsync Resolves Conflicting Options

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 SQL 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 SQL Central.
7. Options specified for the publication, whether by a SQL statement or in SQL Central.

> **i 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.

1.6.3.2 BufferDownload (bd) Extended Option

Specifies whether the entire download from the MobiLink server should be read into the cache before applying it to the remote database.

Syntax

```
bd={ON | OFF}; ...
```

```
BufferDownload={ON | OFF}; ...
```

Remarks

The default is ON. The default results in lower load on the MobiLink server and using it should improve server-side throughput.

When BufferDownload is set to off, dbmlsync applies the download as it is read.

1.6.3.3 CommunicationAddress (adr) Extended Option

Specifies network protocol options for connecting to the MobiLink server.

Syntax

```
adr=protocol-option; ...
```

```
CommunicationAddress=protocol-option; ...
```

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.

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

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'
```

Related Information

The Relay Server Configuration File
MobiLink Client Network Protocol Options [page 24]
CommunicationType (ctp) Extended Option [page 167]

1.6.3.4 CommunicationType (ctp) Extended Option

Specifies the type of network protocol to use for connecting to the MobiLink server.

Syntax

```
ctp=network-protocol; ...
```

```
CommunicationType=network-protocol; ...
```

Remarks

`network-protocol` can be one of *tcpip*, *tls*, *http*, or *https*. The default is *tcpip*.

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

Example

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

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

The following SQL illustrates how to store this option in the database.

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION ctp='tcpip'
```

Related Information

MobiLink Client/Server Communications Encryption
CommunicationAddress (adr) Extended Option [page 166]

1.6.3.5 ConflictRetries (cr) Extended Option

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

Syntax

```
cr=number; ...
```

```
ConflictRetries=number; ...
```

Remarks

When tables are not locked during synchronization, it is possible for operations to be applied to the database between the time the upload is built and the time that the download is applied. If these changes affect rows that are also changed by the download, dbmlsync considers this to be a conflict and does not apply the download stream. When this occurs dbmlsync retries the entire synchronization. Normally the synchronization succeeds on the next attempt but it is possible for a new conflict to force a new retry. This option controls the maximum number of retries that are performed.

This option is useful only if the LockTables option is OFF, which is the default.

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

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';
```

Related Information

Conflict Handling Overview

1.6.3.6 ContinueDownload (cd) Extended Option

Restarts a previously failed download.

Syntax

```
cd={ ON | OFF }; ...
```

```
ContinueDownload={ ON | OFF }; ...
```

Remarks

This option can only be used if the -kpd dbmlsync option or the KeepPartialDownload synchronization profile option was specified on the synchronization that failed.

If dbmlsync does not receive the entire download from the server, dbmlsync does not apply any of the download data to the remote database. However, if the -kpd dbmlsync option was specified on the failed download, dbmlsync stores the part of the download it did receive in a temporary file on the remote device, so that it can be restarted later. When you set the extended option cd=on, dbmlsync restarts the download and attempts to download the part of the previous download that it did not receive. If it is able to download the remaining data, it applies the complete download to your remote database.

If there is any new data to be uploaded when you set cd=on, the synchronization fails without restarting the download. If the download cannot be restarted, synchronization fails.

You can also restart a download with the -dc option or with the sp_hook_dbmlsync_end hook.

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';
```

Related Information

Resumption of Failed Downloads
-kpd dbmlsync Option [page 137]
sp_hook_dbmlsync_set_extended_options [page 285]
-dc dbmlsync Option [page 130]
sp_hook_dbmlsync_end [page 265]

1.6.3.7 DisablePolling (p) Extended Option

Disables automatic logscan polling.

Syntax

```
p={ON | OFF}; ...
```

```
DisablePolling={ON | OFF}; ...
```

Remarks

To build an upload, dbmlsync must scan the transaction log. Usually it does this just before synchronization. However, when synchronizations are scheduled or the sp_hook_dbmlsync_delay hook is used, dbmlsync by default scans the log in the time between synchronizations. This behavior is more efficient because the log is already at least partially scanned when synchronization begins. This default behavior is called logscan polling.

Logscan polling is on by default but only has an effect when synchronizations are scheduled or when sp_hook_dbmlsync_delay hook is used. When in effect, polling occurs at set intervals: dbmlsync scans to the end of the log, waits for the polling period, and then scans any new transactions in the log. By default, the polling period is 1 minute, but it can be changed with the dbmlsync -pp option or the PollingPeriod extended option.

The default is to not disable logscan polling (*OFF*).

This option is identical to dbmlsync *-p*.

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';
```

Related Information

PollingPeriod (pp) Extended Option [page 185]
-p dbmlsync Option [page 142]
-pp dbmlsync Option [page 145]

1.6.3.8 DownloadOnly (ds) Extended Option

Specifies that synchronization should be download-only.

Syntax

```
ds={ ON | OFF }; ...
```

```
DownloadOnly={ ON | OFF }; ...
```

Remarks

When download-only synchronization occurs, dbmlsync does not upload any row operations or data. However, it does upload information about the schema and progress offset.

In addition, dbmlsync ensures that changes on the remote database that have not been uploaded are not overwritten during download-only synchronization. It does this by scanning the log to detect rows with operations waiting to be uploaded. If any of these rows is modified by the download, the download is rolled back and the synchronization fails. If the synchronization fails for this reason, you must do a full synchronization to correct the problem.

When you have remotes that are synchronized by download-only synchronization, you should regularly do a full synchronization to reduce the amount of log that is scanned by the download-only synchronization. Otherwise, the download-only synchronizations take an increasingly long time to complete. If this is a problem, you can alternatively use a download-only publication to avoid log issues during synchronization.

The default is *OFF* (perform both upload and download).

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';
```

Related Information

Scripts Required for Synchronization
Download-only Publications [page 90]
Upload-only and Download-only Synchronizations
-ds dbmlsync Option [page 133]

1.6.3.9 DownloadReadSize (drs) Extended Option

For restartable downloads, specifies the maximum amount of data that may need to be re-sent after a communications failure.

Syntax

```
drs=number[ K ]; ...
```

```
DownloadReadSize=number[ K ]; ...
```

Remarks

The DownloadReadSize option is only useful when doing restartable downloads.

The download read size is specified in units of bytes. Use the suffix k to optionally specify units of kilobytes.

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 ranges between 0 and the DownloadReadSize -1. So for example, if the DownloadReadSize is 100 bytes and an error occurs after reading 497 bytes, the last 97 bytes read are lost. Bytes that are lost in this way are re-sent when the download is restarted.

In general, larger DownloadReadSize values result in better performance on successful synchronizations but result in more data being re-sent 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.

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';
```

Related Information

Resumption of Failed Downloads
-drs dbmlsync Option [page 132]
ContinueDownload (cd) Extended Option [page 169]
sp_hook_dbmlsync_end [page 265]
-dc dbmlsync Option [page 130]

1.6.3.10 ErrorLogSendLimit (el) Extended Option

Specifies how much of the remote message log file dbmlsync should send to the server when synchronization error occurs.

Syntax

```
el=number[ K | M ]; ...
```

```
ErrorLogSendLimit=number[ K | M ]; ...
```

Remarks

This option is specified in units of bytes. Use the suffix k or m to specify units of kilobytes or megabytes, respectively.

This option specifies the number of bytes of the message log that dbmlsync sends to the MobiLink server when errors occur during synchronization. Set this option to *0* if you don't want any dbmlsync message log to be sent.

The default is *32K*.

When this option is non-zero, the error log is uploaded when a client-side error occurs. Not all client-side errors cause the log to be sent: the log is not sent for communication errors or errors that occur when dbmlsync is not connected to the MobiLink server. If the error occurs after the upload is sent, the error log is uploaded only if the SendDownloadAck extended option is set to ON.

If ErrorLogSendLimit is set to be large enough, dbmlsync sends the entire message log from the current session to the MobiLink server. For example, if the message log messages were appended to an old message log file, dbmlsync only sends the new messages generated in the current session. If the total length of new messages is greater than ErrorLogSendLimit, dbmlsync only uploads the messages log up to the specified size.

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".

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';
```

Related Information

-v dbmlsync Option [page 157]
-e dbmlsync Option [page 134]
-eu dbmlsync Option [page 136]
Verbose (v) Extended Option [page 195]
VerboseHooks (vs) Extended Option [page 196]
VerboseMin (vm) Extended Option [page 197]
VerboseOptions (vo) Extended Option [page 198]
VerboseRowCounts (vn) Extended Option [page 199]
VerboseRowValues (vr) Extended Option [page 200]
VerboseUpload (vu) Extended Option [page 201]

1.6.3.11 FireTriggers (ft) Extended Option

Specifies that triggers should be fired on the remote database when the download is applied.

Syntax

```
ft={ ON | OFF }; ...
```

```
FireTriggers={ ON | OFF }; ...
```

Remarks

The default is *ON*.

Example

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

```
dbmlsync -e "ft=off"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION ft='off';
```

1.6.3.12 HoverRescanThreshold (hrt) Extended Option

When you are using scheduling, this limits the amount of discarded memory that is allowed to accumulate before a rescan is performed.

Syntax

```
hrt=number[ K | M ]; ...
```

```
HoverRescanThreshold=number[ K | M ]; ...
```

Remarks

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

When more than one -n option or -s option is specified in the command line, dbmlsync may experience fragmentation which results in discarded memory. The discarded memory can only be recovered by rescanning the database transaction log. This option lets you specify a limit on the amount of discarded memory that is allowed to accumulate before the log is rescanned and the memory recovered. Another way to control the recovery of discarded memory is to implement the sp_hook_dbmlsync_log_rescan stored procedure.

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';
```

Related Information

sp_hook_dbmlsync_log_rescan [page 268]

1.6.3.13 IgnoreHookErrors (eh) Extended Option

Specifies that errors that occur in hook functions should be ignored.

Syntax

```
eh={ ON | OFF }; ...
```

```
IgnoreHookErrors={ ON | OFF }; ...
```

Remarks

The default is *OFF*.

This option is equivalent to the dbmlsync -eh option.

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';
```

1.6.3.14 IgnoreScheduling (isc) Extended Option

Specifies that the Schedule extended option should be ignored.

Syntax

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

```
IgnoreScheduling={ ON | OFF }; ...
```

Remarks

If set to ON, dbmlsync ignores the Schedule extended option and synchronizes immediately. The default is *OFF*.

This option is equivalent to the dbmlsync -is option.

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';
```

Related Information

Synchronization Schedules [page 110]
Schedule (sch) Extended Option [page 186]

1.6.3.15 Increment (inc) Extended Option

Enables incremental uploads and controls the size of upload increments.

Syntax

```
inc=number[ K | M ]; ...
```

```
Increment=number[ K | M ]; ...
```

Remarks

The value of this option specifies, very approximately, the size of each upload part in bytes. Use the suffix k or m to specify units of kilobytes or megabytes, respectively.

When this option is set to a non-zero value, uploads are sent to MobiLink in one or more parts. This could be useful if dbmlsync has difficulty maintaining a connection to the MobiLink server for long enough to complete the full upload. the default is 0.

The value of the option controls the size of each upload part as follows. Dbmlsync builds the upload by scanning the database transaction log. When this option is set, dbmlsync scans the number of bytes that are set in the option and then continues scanning to the first point at which there are no outstanding partial transactions; the next point at which all transactions have either been committed or rolled back. It then sends what it has scanned as an upload part and resumes scanning the log from where it left off.

You cannot use the Increment extended option with scripted upload or transactional upload.

Example

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

```
dbmlsync -e "inc=32000"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION inc='32k';
```

1.6.3.16 LockTables (lt) Extended Option

Specifies that tables in the publications being synchronized should be locked before synchronizing.

Syntax

```
lt={ ON | OFF | SHARE | EXCLUSIVE }; ...
```

```
LockTables={ ON | OFF | SHARE | EXCLUSIVE }; ...
```

Remarks

SHARE means that dbmlsync locks all synchronization tables in shared mode. EXCLUSIVE means that dbmlsync locks all synchronization tables in exclusive mode. For all platforms, ON is the same as SHARE.

The default is OFF. By default, dbmlsync does not lock any synchronization tables except for the following situations:

- If there is a publication that uses script-based upload in the current synchronization or if there is an sp_hook_dbmlsync_schema_upgrade hook defined in the remote database, dbmlsync locks the synchronization tables with SHARE.

Set to ON to prevent modifications during synchronization.

When synchronization tables are locked in exclusive mode, 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.

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';
```

Related Information

How Locking Works
Concurrency During Synchronization [page 108]
Event Hooks for SQL Anywhere Clients [page 232]
LOCK TABLE Statement

1.6.3.17 MirrorLogDirectory (mld) Extended Option

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

Syntax

```
mld=dir; ...
```

```
MirrorLogDirectory=dir; ...
```

Remarks

This option makes it possible for dbmlsync to delete old transaction log mirror files when either of the following two circumstances occur:

- the offline transaction log mirror is located in a different directory from the transaction log mirror
  or
- dbmlsync is run on a different computer from the remote database server

In a normal setup, the active transaction log mirror and renamed transaction log mirror files are located in the same directory, and dbmlsync is run on the same computer as the remote database, so this option is not required and old transaction log mirror files are automatically deleted.

Transaction logs in this directory are only affected if the delete_old_logs database option is set to On, Delay, or `n` days.

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';
```

Related Information

delete_old_logs Option [MobiLink][SQL Remote]

1.6.3.18 MobiLinkPwd (mp) Extended Option

Specifies the MobiLink password.

Syntax

```
mp=password; ...
```

```
MobiLinkPwd=password; ...
```

Remarks

Specifies the password used to connect to the MobiLink server. This password should be the correct password for the MobiLink user whose subscriptions are being synchronized. The default is null.

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

Example

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

```
dbmlsync -e "mp=password"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION mp='password';
```

Related Information

NewMobiLinkPwd (mn) Extended Option [page 182]
-mn dbmlsync Option [page 138]
-mp dbmlsync Option [page 139]

1.6.3.19 NewMobiLinkPwd (mn) Extended Option

Specifies a new MobiLink password.

Syntax

```
mn=new-password; ...
```

```
NewMobiLinkPwd=new-password; ...
```

Remarks

Specifies a new password for the MobiLink user whose subscriptions are being synchronized. Use this option when you want to change an existing password. The default is not to change the password.

Example

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

```
dbmlsync -e "mp=oldpassword;mn=newpassword"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION mp='oldpassword';mn='newpassword'
```

Related Information

MobiLinkPwd (mp) Extended Option [page 181]
-mn dbmlsync Option [page 138]
-mp dbmlsync Option [page 139]

1.6.3.20 NoSyncOnStartup (nss) Extended Option

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

Syntax

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

```
NoSyncOnStartup={ on | off }; ...
```

Remarks

This option has an effect only when the schedule extended option is used with the EVERY or INFINITE clause. These scheduling options cause dbmlsync to automatically synchronize on startup.

The default is off.

When you set NoSyncOnStartup to on and use a schedule with the INFINITE clause, a synchronization does not occur until a window message is received.

When you set NoSyncOnStartup to on and use a schedule with the EVERY clause, the first synchronization after startup occurs after the amount of time specified in the EVERY clause.

This setting does not affect the behavior of the schedule in any way other than at dbmlsync startup.

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';
```

Related Information

Synchronization Schedules [page 110]
Schedule (sch) Extended Option [page 186]

1.6.3.21 OfflineDirectory (dir) Extended Option

Specifies the path containing offline transaction logs.

Syntax

```
dir=path; ...
```

```
OfflineDirectory=path; ...
```

Remarks

By default, dbmlsync checks for renamed logs in the same directory as the online transaction log. This option can be specified if the renamed offline transaction logs are located in a different directory.

Do not use this option if the renamed (offline) transaction logs are in the same directory as the online transaction log. This configuration provides more flexibility because it allows you to move the database and transaction log files (for example, in a cloud or by using dblog -t) without having to change the OfflineDirectory setting. However, there is a slight performance penalty because the database server must do more work to retrieve the pages, so if performance is critical, using the OfflineDirectory option may be best for your deployment.

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';
```

1.6.3.22 PollingPeriod (pp) Extended Option

Specifies the logscan polling period.

Syntax

```
pp=number[S | M | H | D ]; ...
```

```
PollingPeriod=number[S | M | H | D ]; ...
```

Remarks

This option specifies the interval between log scans. Use the suffix s, m, h, or d to specify seconds, minutes, hours or days, respectively. The default is *1* minute. If you do not specify a suffix, the default unit of time is minutes.

Logscan polling occurs only when you are scheduling synchronizations or using the sp_hook_dbmlsync_delay hook.

This option is identical to dbmlsync *-pp*.

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';
```

Related Information

DisablePolling (p) Extended Option [page 170]
-pp dbmlsync Option [page 145]
-p dbmlsync Option [page 142]
sp_hook_dbmlsync_delay [page 250]
Schedule (sch) Extended Option [page 186]

1.6.3.23 Schedule (sch) Extended Option

Specifies a schedule for synchronization.

Syntax

```
sch=schedule; ...
```

```
Schedule=schedule; ...
```

```
schedule : { EVERY:hhhh:mm | INFINITE | singleSchedule }
```

```
hhhh : 00 ... 9999
```

```
mm : 00 ... 59
```

```
singleSchedule : day @hh:mm[ AM | PM ] [ -hh:mm[ AM | PM ] ] ,...
```

```
hh : 00 ... 24
```

```
mm : 00 ... 59
```

```
day :
 EVERYDAY | WEEKDAY | MON | TUE | WED | THU | FRI | SAT | SUN | dayOfMonth
```

```
dayOfMonth : 0... 31
```

Remarks

EVERY

The EVERY keyword causes synchronization to occur on startup, and then repeat indefinitely after the specified time period. If the synchronization process takes longer than the specified period, synchronization starts again immediately.

To avoid having a synchronization occur when dbmlsync starts, use the extended option NoSyncOnStartup.

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. You may also use the full forms of the day, such as Monday. You must use the full forms of the day names if the language you are using is not English, is not the language requested by the client in the connection string, and is not the language which appears in the server messages window.

dayOfMonth

To specify the last day of the month regardless of the length of the month, set the `dayOfMonth` to 0.

INFINITE

The INFINITE keyword causes dbmlsync to synchronize on startup, and then not to synchronize again until synchronization is initiated by another program sending a window message to dbmlsync. You can use the dbmlsync extended option NoSyncOnStartup to avoid the initial synchronization.

You can use this option in conjunction with the dbmlsync -wc option to wake up dbmlsync and perform a synchronization.

If a previous synchronization is still incomplete at a scheduled time, the scheduled synchronization commences when the previous synchronization completes.

The default is no schedule.

The Schedule option is ignored when the Dbmlsync API is used or when the SQL SYNCHRONIZE statement is used.

The IgnoreScheduling extended option and the dbmlsync -is option instruct dbmlsync to ignore scheduling, so that synchronization is immediate.

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';
```

Related Information

Synchronization Schedules [page 110]
NoSyncOnStartup (nss) Extended Option [page 183]
-wc dbmlsync Option [page 159]
IgnoreScheduling (isc) Extended Option [page 177]
-is dbmlsync Option [page 136]

1.6.3.24 ScriptVersion (sv) Extended Option

Specifies a script version.

⚠ Caution

It is strongly recommended that you specify the script version using the SCRIPT VERSION clause on the CREATE SYNCHRONIZATION SUBSCRIPTION and ALTER SYNCHRONIZATION SUBSCRIPTION statements instead of using the ScriptVersion extended option because using the SCRIPT VERSION clause greatly simplifies the problem of doing schema upgrades.

The ScriptVersion (sv) extended option overrides the value stored using the SCRIPT VERSION clause and should only be used for backward compatibility purposes or for the rare case where you need to explicitly specify the script version used for a synchronization.

Syntax

```
sv=version-name; ...
```

```
ScriptVersion=version-name; ...
```

Remarks

The script version determines which scripts are run by MobiLink on the consolidated database during synchronization. The default script version name is *default*.

Example

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

```
dbmlsync -e "sv=SysAd001"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION sv='SysAd001';
```

Related Information

CREATE SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink]
ALTER SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink]

1.6.3.25 SendDownloadAck (sa) Extended Option

Specifies that a download acknowledgement should be sent from the client to the server.

Syntax

```
sa={ ON | OFF }; ...
```

```
SendDownloadAck={ ON | OFF }; ...
```

Remarks

A download acknowledgement lets MobiLink server know for sure that a download has been applied to a remote database. You can write synchronization scripts in your consolidated database to handle the acknowledgement and perform business logic at the time of the acknowledgement. A download acknowledgement may not be sent if the network session is dropped after the client applies the download, so your scripts should allow for this possibility.

Note: When SendDownloadAck is set to ON and you are in verbose mode, an acknowledgement line is written to the client log.

The default is *OFF*.

Example

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

```
dbmlsync -e "sa=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION sa='on';
```

Related Information

nonblocking_download_ack Connection Event

1.6.3.26 SendTriggers (st) Extended Option

Specifies that trigger actions should be sent on upload.

Syntax

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

```
SendTriggers={ ON | OFF }; ...
```

Remarks

Cascaded deletes are also considered trigger actions.

The default is *OFF*.

If two subscriptions both contain one or more of the same tables, then both subscriptions must be synchronized using the same setting for the SendTriggers option.

i Note

Trigger actions that occur as a result of database changes made as part of the download phase of synchronization are never synchronized, regardless of the value of the SendTriggers option.

In order for an operation in a trigger to be synchronized when the SendTrigger option is set to *ON*, the operation in the trigger must occur on a table in a publication that is being synchronized. It is not necessary for the base operation that caused the trigger to fire to be in the publication as well.

Dbmlsync coalesces operations that occur on a single row, but if the SendTrigger option is set to *ON*, then all the operations that fire in the trigger are also sent, even if the base operation that caused the trigger to fire is coalesced into another operation.

Built in referential integrity actions on foreign keys (ON DELETE CASCADE and ON UPDATE CASCADE) are considered trigger actions, and will not synchronize unless the SendTrigger option is set to *ON*. The one exception is the case where a referential integrity action takes place on a row that is already in the upload stream. In this case, the referential integrity action in the system generated trigger is coalesced with the state of the row that is already in the upload stream, as illustrated in the following example.

In this example, there is a foreign key between the **Parent** table and the **Child** table, with a referential integrity action of ON DELETE CASCADE.

```
INSERT INTO Parent (pid, pname) values ( 100, 'Amy' );
INSERT INTO Child (cid, pid, cname) values ( 2000, 100, 'Alex' );
COMMIT;
DELETE FROM Parent WHERE pid = 100;
COMMIT;
```

It is important to be aware that the RI action also deletes row 2000 from the **Child** table when you delete row 100 from the **Parent** table. If dbmlsync were run at this point, the INSERT and then DELETE on row 100 of the **Parent** table would result in the row not being synchronized. However, the insert into the **Child** table would be sent if the SendTrigger option is set to *OFF*. In this case, because dbmlsync has already added row 2000 of the **Child** table into the upload stream, the referential integrity action on the delete of row 100 on the **Parent** table, which deletes row 2000 of the **Child** table, is coalesced with the existing row in the upload stream, resulting in neither row being synchronized, and maintaining the consistency of the data at both sides.

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';
```

1.6.3.27 TableOrder (tor) Extended Option

Specifies the order of tables in the upload.

Syntax

```
tor=tables; ...
```

```
TableOrder=tables; ...
```

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

Remarks

This option allows you to specify the order in which tables are uploaded. You must specify all tables that are to be uploaded. If you include tables that are not included in the synchronization, they are ignored.

The table order that you specify must ensure referential integrity. 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.

Example

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

```
dbmlsync -e "tor=admin,primary,foreign"
```

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,primary,foreign;
```

Related Information

How the Upload Is Processed
Referential Integrity and Synchronization
TableOrderChecking (toc) Extended Option [page 193]

1.6.3.28 TableOrderChecking (toc) Extended Option

Lets you disable referential integrity checking on the table order specified by the TableOrder extended option.

Syntax

```
toc={ OFF | ON }; ...
```

```
TableOrderChecking={ OFF | ON }; ...
```

Remarks

In most applications, tables on the remote and consolidated databases have the same foreign key relationships. In these cases, you should leave TableOrderChecking at its default value of ON, and dbmlsync ensures that no table is uploaded before another table on which it has a foreign key. This ensures referential integrity.

This option is useful when the consolidated and remote databases have different foreign key relationships. Use it with the TableOrder extended option to specify an order of tables that satisfies the referential integrity constraints on the server even though it may violate those present on the remote.

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

The default is *ON*.

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';
```

Related Information

How the Upload Is Processed
Referential Integrity and Synchronization
TableOrder (tor) Extended Option [page 192]

1.6.3.29 UploadOnly (uo) Extended Option

Specifies that synchronization should only include an upload.

Syntax

```
uo={ ON | OFF }; ...
```

```
UploadOnly={ ON | OFF }; ...
```

Remarks

During upload-only synchronization, dbmlsync prepares and sends an upload to the MobiLink server exactly as in a normal full synchronization. However, instead of sending a download back down, the MobiLink server sends only an acknowledgement indicating if the upload was successfully committed.

For a list of the scripts that must be defined for upload-only synchronization, see the documentation on scripts required for synchronization.

The default is *OFF*.

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';
```

Related Information

Upload-only and Download-only Synchronizations
Scripts Required for Synchronization
DownloadOnly (ds) Extended Option [page 171]

1.6.3.30 Verbose (v) Extended Option

Specifies full verbosity.

Syntax

```
v={ ON | OFF }; ...
```

```
Verbose={ ON | OFF }; ...
```

Remarks

This option specifies a high level of verbosity, which may affect performance and should normally be used in the development phase only.

This option is identical to dbmlsync *-v+*. If you specify both -v and the extended options and there are conflicts, the -v option overrides the extended option. If there is no conflict, the verbosity logging options are additive and 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.

The default is *OFF*.

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';
```

Related Information

-v dbmlsync Option [page 157]
VerboseHooks (vs) Extended Option [page 196]
VerboseMin (vm) Extended Option [page 197]
VerboseOptions (vo) Extended Option [page 198]
VerboseRowCounts (vn) Extended Option [page 199]
VerboseRowValues (vr) Extended Option [page 200]
VerboseUpload (vu) Extended Option [page 201]

1.6.3.31 VerboseHooks (vs) Extended Option

Specifies that messages related to hook scripts should be logged.

Syntax

```
vs={ ON | OFF }; ...
```

```
VerboseHooks={ ON | OFF }; ...
```

Remarks

This option is identical to 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 and 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.

The default is *OFF*.

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';
```

Related Information

Event Hooks for SQL Anywhere Clients [page 232]
-v dbmlsync Option [page 157]
Verbose (v) Extended Option [page 195]
VerboseMin (vm) Extended Option [page 197]
VerboseOptions (vo) Extended Option [page 198]
VerboseRowCounts (vn) Extended Option [page 199]
VerboseRowValues (vr) Extended Option [page 200]
VerboseUpload (vu) Extended Option [page 201]

1.6.3.32 VerboseMin (vm) Extended Option

Specifies that a small amount of information should be logged.

Syntax

```
vm={ ON | OFF }; ...
```

```
VerboseMin={ ON | OFF }; ...
```

Remarks

This option is identical to 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 and 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.

The default is *OFF*.

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';
```

Related Information

-v dbmlsync Option [page 157]
Verbose (v) Extended Option [page 195]
VerboseOptions (vo) Extended Option [page 198]
VerboseRowCounts (vn) Extended Option [page 199]
VerboseRowValues (vr) Extended Option [page 200]
VerboseUpload (vu) Extended Option [page 201]

1.6.3.33 VerboseOptions (vo) Extended Option

Specifies that information should be logged about the command line options (including extended options) that you have specified.

Syntax

```
vo={ ON | OFF }; ...
```

```
VerboseOptions={ ON | OFF }; ...
```

Remarks

This option is identical to 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 and 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.

The default is *OFF*.

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';
```

Related Information

-v dbmlsync Option [page 157]
Verbose (v) Extended Option [page 195]
VerboseMin (vm) Extended Option [page 197]
VerboseRowCounts (vn) Extended Option [page 199]
VerboseRowValues (vr) Extended Option [page 200]
VerboseUpload (vu) Extended Option [page 201]

1.6.3.34 VerboseRowCounts (vn) Extended Option

Specifies that the number of rows that are uploaded and downloaded should be logged.

Syntax

```
vn={ ON | OFF }; ...
```

```
VerboseRowCounts={ ON | OFF }; ...
```

Remarks

This option is identical to dbmlsync *-vn*. If you specify both -v and the extended options and there are conflicts, the -v option overrides the extended option. If there is no conflict, the verbosity logging options are additive and 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.

The default is *OFF*.

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';
```

Related Information

-v dbmlsync Option [page 157]
Verbose (v) Extended Option [page 195]
Verbose (v) Extended Option [page 195]
VerboseMin (vm) Extended Option [page 197]
VerboseOptions (vo) Extended Option [page 198]
VerboseRowValues (vr) Extended Option [page 200]
VerboseUpload (vu) Extended Option [page 201]

1.6.3.35 VerboseRowValues (vr) Extended Option

Specifies that the values of rows that are uploaded and downloaded should be logged.

Syntax

```
vr={ ON | OFF }; ...
```

```
VerboseRowValues={ ON | OFF }; ...
```

Remarks

This option is identical to dbmlsync *-vr*. If you specify both -v and the extended options and there are conflicts, the -v option overrides the extended option. If there is no conflict, the verbosity logging options are additive and 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.

The default is *OFF*.

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';
```

Related Information

-v dbmlsync Option [page 157]
Verbose (v) Extended Option [page 195]
Verbose (v) Extended Option [page 195]
VerboseMin (vm) Extended Option [page 197]
VerboseOptions (vo) Extended Option [page 198]
VerboseRowCounts (vn) Extended Option [page 199]
VerboseUpload (vu) Extended Option [page 201]

1.6.3.36 VerboseUpload (vu) Extended Option

Specifies that information about the upload steam should be logged.

Syntax

```
vu={ ON | OFF }; ...
```

```
VerboseUpload={ ON | OFF }; ...
```

Remarks

This option is identical to dbmlsync *-vu*. If you specify both -v and the extended options and there are conflicts, the -v option overrides the extended option. If there is no conflict, the verbosity logging options are additive and 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.

The default is *OFF*.

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';
```

Related Information

-v dbmlsync Option [page 157]
Verbose (v) Extended Option [page 195]
Verbose (v) Extended Option [page 195]
VerboseMin (vm) Extended Option [page 197]
VerboseOptions (vo) Extended Option [page 198]
VerboseRowCounts (vn) Extended Option [page 199]
VerboseRowValues (vr) Extended Option [page 200]

1.6.4 MobiLink SQL Statements

The following are the SQL statements used for configuring and running MobiLink SQL Anywhere clients:

- ALTER PUBLICATION statement [MobiLink] [SQL Remote]
- ALTER SYNCHRONIZATION PROFILE statement [MobiLink]
- ALTER SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]
- ALTER SYNCHRONIZATION USER statement [MobiLink]
- CREATE PUBLICATION statement [MobiLink] [SQL Remote]
- CREATE SYNCHRONIZATION PROFILE statement [MobiLink]
- CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]
- CREATE SYNCHRONIZATION USER statement [MobiLink]
- DROP PUBLICATION statement [MobiLink] [SQL Remote]
- DROP SYNCHRONIZATION PROFILE statement [MobiLink]
- DROP SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]
- DROP SYNCHRONIZATION USER statement [MobiLink]
- GRANT ROLE statement
- REVOKE ROLE statement
- START SYNCHRONIZATION DELETE statement [MobiLink]

- START SYNCHRONIZATION SCHEMA CHANGE statement [MobiLink]
- STOP SYNCHRONIZATION DELETE statement [MobiLink]
- STOP SYNCHRONIZATION SCHEMA CHANGE statement [MobiLink]
- SYNCHRONIZE statement [MobiLink]

UltraLite clients

See the UltraLite documentation for information about UltraLite SQL statements.

Related Information

UltraLite SQL Statements
ALTER PUBLICATION Statement [MobiLink] [SQL Remote]
ALTER SYNCHRONIZATION PROFILE Statement [MobiLink]
ALTER SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink]
ALTER SYNCHRONIZATION USER Statement [MobiLink]
CREATE PUBLICATION Statement [MobiLink] [SQL Remote]
CREATE SYNCHRONIZATION PROFILE Statement [MobiLink]
CREATE SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink]
CREATE SYNCHRONIZATION USER Statement [MobiLink]
DROP PUBLICATION Statement [MobiLink] [SQL Remote]
DROP SYNCHRONIZATION PROFILE Statement [MobiLink]
DROP SYNCHRONIZATION SUBSCRIPTION Statement [MobiLink]
DROP SYNCHRONIZATION USER Statement [MobiLink]
GRANT ROLE Statement
REVOKE ROLE Statement
START SYNCHRONIZATION DELETE Statement [MobiLink]
START SYNCHRONIZATION SCHEMA CHANGE Statement [MobiLink]
STOP SYNCHRONIZATION DELETE Statement [MobiLink]
STOP SYNCHRONIZATION SCHEMA CHANGE Statement [MobiLink]
SYNCHRONIZE Statement [MobiLink]

1.6.5 MobiLink Synchronization Profiles

Synchronization profiles allow you to place some dbmlsync options in the database. The synchronization profile you create can contain a variety of synchronizations options.

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

- CREATE SYNCHRONIZATION PROFILE statement [MobiLink]
- ALTER SYNCHRONIZATION PROFILE statement [MobiLink]
- DROP SYNCHRONIZATION PROFILE statement [MobiLink]

Synchronization profiles can be accessed using the dbmlsync -sp option, the Sync method in the Dbmlsync API, the SQL SYNCHRONIZE statement and the remote tasks created from central administration of remote databases. In all cases there is the ability to specify additional options that are merged with the options in the synchronization profiles. If any of the extra option specified conflicts with an option specified in the synchronization profile, the value specified by the extra options is used.

When using the dbmlsync -sp options, all the other options on the command line are treated as extra options. The other interfaces provide a specific parameter or clause for specifying extra options.

The following options can be specified in a synchronization profile:

Long option name	Short name	Allowed values	Description
AuthParms	*ap*	*String*	Supplies parameters to the authenticate_parameters script and to authentication parameters.
ApplyDnldFile	*ba*	*String*	Applies a download file.
Background	*bk*	*String*	Enables background synchronization when set to TRUE.
BackgroundRetry	*bkr*	*Integer*	Controls how dbmlsync behaves after a background synchronization is interrupted.
ContinueDownload	*dc*	*Boolean*	Restarts a previously failed download.
CreateDnldFile	*bc*	*String*	Creates a download file.
DnldFileExtra	*be*	*String*	When creating a download file, this option specifies an extra string to be included in the file.
DownloadOnly	*ds*	*Boolean*	Performs a download-only synchronization.
DownloadReadSize	*drs*	*Integer*	For restartable downloads, specifies the maximum amount of data that may need to be re-sent after a communications failure.
ExtOpt	*e*	*String*	Specifies extended options.
IgnoreHookErrors	*eh*	*Boolean*	Ignores errors that occur in hook functions.
IgnoreScheduling	*is*	*Boolean*	Ignores scheduling instructions so that synchronization is immediate.
KeepPartialDownload	*kpd*	*Boolean*	In the event of a download failure, saves the information required to attempt restarting the failed download.

Long option name	Short name	Allowed values	Description
KillConnections	*d*	*Boolean*	Drops conflicting locks to the remote database.
LogRenameSize	*x*	An integer optionally followed by K or M	Renames and restarts the transaction log after it has been scanned for upload data.
MobiLinkPwd	*mp*	*String*	Supplies the password of the MobiLink user.
MLUser	*u*	*String*	Specifies the MobiLink user name.
NewMobiLinkPwd	*mn*	*String*	Supplies a new password for the MobiLink user. Use this option when you want to change an existing password.
Ping	*pi*	*Boolean*	Pings a MobiLink server to confirm communications between the client and MobiLink.
Publication	*n*	*String*	This option is deprecated. Specifies the publications(s) to synchronize. Publication can only be specified once in a synchronization profile but the command line option can be specified multiple times.
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.
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.
Subscription	*s*	*String*	Specifies the subscription(s) to synchronize. Subscription can only be specified once in a synchronization profile.
TransactionalUpload	*tu*	*Boolean*	Specifies that each transaction on the remote database should be uploaded as a separate transaction within one synchronization.

Long option name	Short name	Allowed values	Description
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.
UploadOnly	*uo*	*Boolean*	Specifies that synchronization only includes an upload, and that no downloads occur.
UploadRowCnt	*urc*	*Integer*	Specifies an estimate of the number of rows to be uploaded in a synchronization.
Verbosity		*String* (a comma-separated list of options)	Controls dbmlsync verbosity. Similar to the Verbose MobiLink SQL Anywhere client extended option. 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 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

In this section:

Restarts a previously failed download.

CreateDnldFile Synchronization Profile Option [page 212]
: Creates a download file with the specified name.

DnldFileExtra Synchronization Profile Option [page 213]
: When creating a download file, this option specifies an extra string to be included in the file.

DownloadOnly Synchronization Profile Option [page 214]
: Perform a download-only synchronization when set to on.

DownloadReadSize Synchronization Profile Option [page 215]
: For restartable downloads, specifies the maximum number of bytes that may need to be re-sent after a communications failure.

ExtOpt Synchronization Profile Option [page 216]
: Specifies extended options.

IgnoreHookErrors Synchronization Profile Option [page 217]
: Ignore errors that occur in hook functions when set to on.

IgnoreScheduling Synchronization Profile Option [page 217]
: Ignores the Schedule extended option so that synchronization is immediate.

KeepPartialDownload Synchronization Profile Option [page 218]
: Saves the information required to attempt restarting the failed download.

KillConnections Synchronization Profile Option [page 219]
: Drops conflicting locks to the remote database.

LogRenameSize Synchronization Profile Option [page 220]
: Renames and restarts the transaction log.

MobiLinkPwd Synchronization Profile Option [page 220]
: Supplies the password of the MobiLink user.

MLUser Synchronization Profile Option (Deprecated) [page 221]
: Specifies the MobiLink user name.

NewMobiLinkPwd Synchronization Profile Option [page 222]
: Supplies a new password for the MobiLink user. Use this option when you want to change the existing password.

Ping Synchronization Profile Option [page 222]
: Pings a MobiLink server.

Publication Synchronization Profile Option (Deprecated) [page 223]
: Specifies the publications(s) to synchronize.

RemoteProgressGreater Synchronization Profile Option [page 224]
: Specifies that the remote offset should be used if it is greater than the consolidated offset. This is equivalent to the -ra option.

RemoteProgressLess Synchronization Profile Option [page 225]
: 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.

Subscription Synchronization Profile Option [page 226]
: Specifies the subscription(s) to synchronize.

TransactionalUpload Synchronization Profile Option [page 227]

Boolean. Specifies that each transaction on the remote database should be uploaded as a separate transaction within one synchronization.

UpdateGenNum Synchronization Profile Option [page 228]
: When creating a download file, this option creates a file that can be used with remote databases that have not yet synchronize. Otherwise, you must perform a synchronization before you apply a download file.

UploadOnly Synchronization Profile Option [page 229]
: Specifies that synchronization only includes an upload, and that no download should occur.

UploadRowCnt Synchronization Profile Option [page 229]
: Integer. Specifies an estimate of the number of rows to be uploaded in a synchronization.

Verbosity Synchronization Profile Option [page 230]
: Controls dbmlsync verbosity.

Related Information

CREATE SYNCHRONIZATION PROFILE Statement [MobiLink]
ALTER SYNCHRONIZATION PROFILE Statement [MobiLink]
DROP SYNCHRONIZATION PROFILE Statement [MobiLink]
UltraLite Synchronization Profile Options

1.6.5.1 AuthParms Synchronization Profile Option

Supplies parameters to the authenticate_parameters script and to authentication parameters.

Syntax

```
ap=parameters
```

```
Authparms=parameters
```

Remarks

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

The parameters are sent to the MobiLink server and passed to the authenticate_parameters script or other events on the consolidated database.

Example

```
CREATE SYNCHRONIZATION PROFILE myprofile 'AuthParms=p1,p2,p3'
```

Related Information

-ap dbmlsync Option [page 122]

1.6.5.2 ApplyDnldFile Synchronization Profile Option

Applies a download file.

Syntax

```
ba=filename
```

```
ApplyDnldFile=filename
```

Remarks

Specify the name of an existing download file to be applied to the remote database.

Example

```
CREATE SYNCHRONIZATION PROFILE myprofile 'ApplyDnldFile=filename'
```

Related Information

-ba dbmlsync Option [page 123]

1.6.5.3 Background Synchronization Profile Option

Enables background synchronization when set to ON.

Syntax

bk= { *ON* | *OFF* }

Background= { *ON* | *OFF* }

Remarks

During a background synchronization, the database server drops the dbmlsync connection to the remote database and rolls back any uncommitted dbmlsync operations if another connection is waiting for access to any database resource that dbmlsync has locked. This allows the other connections to go forward without waiting for the synchronization to complete. Depending on the operations dbmlsync had outstanding when its connection is dropped, there may still be a significant delay for the waiting connection as the database rolls back the dbmlsync uncommitted changes.

Example

The following example shows how to use the Background synchronization profile option:

```
CREATE SYNCHRONIZATION PROFILE myprofile 'Background=on'
```

Related Information

-bk dbmlsync Option [page 125]

1.6.5.4 BackgroundRetry Synchronization Profile Option

Integer. Controls how dbmlsync behaves after a background synchronization is interrupted.

Syntax

bkr=`integer`

BackgroundRetry=`integer`

Remarks

This option has a short form and long form: you can use bkr or BackgroundRetry.

Set this value as an integer greater than or equal to -1. If the value is -1 then dbmlsync retries an interrupted synchronization until it completes, successfully or unsuccessfully, without being interrupted. If the value is 0 then dbmlsync does not retry the interrupted synchronization. If the value is greater than 0 then dbmlsync retries the synchronization up to the number of times specified until it completes. After the specified number of attempts, if the synchronization has not completed, then it is run as a foreground synchronization so it will complete without interruption.

By default BackgroundRetry is 0.

It is an error to set BackgroundRetry to a non-zero value when the Background option has not been set to ON.

This option is ignored when synchronization is initiated using the Dbmlsync API or the SQL SYNCHRONIZE statement.

Example

The following example shows how to use the BackgroundRetry synchronization profile option:

```
CREATE SYNCHRONIZATION PROFILE myprofile 'BackgroundRetry=4'
```

Related Information

-bkr dbmlsync Option [page 126]

1.6.5.5 ContinueDownload Synchronization Profile Option

Restarts a previously failed download.

Syntax

dc= { *ON* | *OFF* }

ContinueDownload= { *ON* | *OFF* }

Remarks

This option can only be used if the -kpd dbmlsync option or the KeepPartialDownload synchronization profile option was specified on the synchronization that failed.

If dbmlsync does not receive the entire download from the server, dbmlsync does not apply any of the download data to the remote database. However, if the -kpd dbmlsync option was specified on the failed download, dbmlsync 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 you specify ContinueDownload=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 ContinueDownload=on, the restartable download fails. You can also restart a failed download using the ContinueDownload extended option or the sp_hook_dbmlsync_end hook.

Example

The following example shows how to use the ContinueDownload synchronization profile option:

```
CREATE SYNCHRONIZATION PROFILE myprofile 'ContinueDownload=on'
```

Related Information

-kpd dbmlsync Option [page 137]
-dc dbmlsync Option [page 130]
sp_hook_dbmlsync_end [page 265]
ContinueDownload (cd) Extended Option [page 169]

1.6.5.6 CreateDnldFile Synchronization Profile Option

Creates a download file with the specified name.

Syntax

```
bc=filename
```

```
CreateDnldFile=filename
```

Remarks

You should use the file extension `.df` for download files.

You can optionally specify a path. If you do not specify a path, the default location is the dbmlsync current working directory, which is the directory where dbmlsync was started.

Optionally, when creating a download file, you can use the -be option to specify a string that can be validated at the remote database, and the -bg option to create a download file for new remote databases.

Example

```
CREATE SYNCHRONIZATION PROFILE myprofile 'CreateDnldFile=dnldl.df'
```

Related Information

MobiLink File-based Download
-bc dbmlsync Option [page 123]

1.6.5.7 DnldFileExtra Synchronization Profile Option

When creating a download file, this option specifies an extra string to be included in the file.

Syntax

be=`string`

DnldFileExtra=`string`

Remarks

The string can be used for authentication or other purposes. It is passed to the sp_hook_dbmlsync_validate_download_file stored procedure on the remote database when the download file is applied.

The string may not contain any semicolons.

Example

The following example shows how to use the DnldFileExtra synchronization profile option:

```
CREATE SYNCHRONIZATION PROFILE myprofile 'DnldFileExtra=val1,val2,val3'
```

Related Information

-be dbmlsync Option [page 124]
sp_hook_dbmlsync_validate_download_file [page 298]

1.6.5.8 DownloadOnly Synchronization Profile Option

Perform a download-only synchronization when set to on.

Syntax

ds= { *ON* | *OFF* }

DownloadOnly= { *ON* | *OFF* }

Remarks

When download-only synchronization occurs, dbmlsync does not upload any database changes. However, it does upload information about the schema and progress offset.

In addition, dbmlsync ensures that changes on the remote database 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.

Example

The following example shows how to use the DownloadOnly synchronization profile option:

```
CREATE SYNCHRONIZATION PROFILE myprofile 'DownloadOnly=on'
```

Related Information

-ds dbmlsync Option [page 133]
DownloadOnly (ds) Extended Option [page 171]

1.6.5.9 DownloadReadSize Synchronization Profile Option

For restartable downloads, specifies the maximum number of bytes that may need to be re-sent after a communications failure.

Syntax

```
drs=size
```

```
DownloadReadSize=size
```

Remarks

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 re-sent 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 re-sent 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.

Example

The following example shows how to use the DownloadReadSize synchronization profile option:

```
CREATE SYNCHRONIZATION PROFILE myprofile 'DownloadReadSize=100'
```

Related Information

-drs dbmlsync Option [page 132]
DownloadReadSize (drs) Extended Option [page 172]

1.6.5.10 ExtOpt Synchronization Profile Option

Specifies extended options.

Syntax

```
e={option=value; ...}
```

```
ExtOpt={option=value; ...}
```

Remarks

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

Extended options are specified in option=value pairs. The list of extended options being set must be enclosed in curly braces {}.

Example

The following example shows how to use the ExtOpt synchronization profile option:

```
CREATE SYNCHRONIZATION PROFILE myprofile
'ExtOpt={lt=exclusive;tableorder=t1,t2,t3}'
```

Related Information

MobiLink SQL Anywhere Client Extended Options [page 160]
-e dbmlsync Option [page 134]

1.6.5.11 IgnoreHookErrors Synchronization Profile Option

Ignore errors that occur in hook functions when set to on.

Syntax

eh= { *ON* | *OFF* }

IgnoreHookErrors= { *ON* | *OFF* }

Example

The following example shows how to use the IgnoreHookErrors synchronization profile option:

```
CREATE SYNCHRONIZATION PROFILE myprofile 'IgnoreHookErrors=on'
```

Related Information

-eh dbmlsync Option [page 135]

1.6.5.12 IgnoreScheduling Synchronization Profile Option

Ignores the Schedule extended option so that synchronization is immediate.

Syntax

is= { *ON* | *OFF* }

IgnoreScheduling= { *ON* | *OFF* }

Remarks

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

Example

The following example shows how to use the IgnoreScheduling synchronization profile option:

```
CREATE SYNCHRONIZATION PROFILE myprofile 'IgnoreScheduling=on'
```

Related Information

Synchronization Schedules [page 110]
-is dbmlsync Option [page 136]

1.6.5.13 KeepPartialDownload Synchronization Profile Option

Saves the information required to attempt restarting the failed download.

Syntax

```
kpd={ TRUE | FALSE }
```

```
KeepPartialDownload={ TRUE | FALSE }
```

Remarks

This option has a short form and long form: use kpd or KeepPartialDownload.

If neither this option nor the -kpd dbmlsync option is used, then dbmlsync does not save the portion of the download that it received before the synchronization failed and the failed download cannot be restarted.

Restart a failed download by using one of the following methods:

- -dc dbmlsync option
- ContinueDownload extended option
- ContinueDownload synchronization profile option
- the restart parameter in the sp_hook_dbmlsync_end stored procedure

Example

The following example shows how to use the KeepPartialDownload synchronization profile option:

```
CREATE SYNCHRONIZATION PROFILE myprofile 'KeepPartialDownload=TRUE';
```

Related Information

-dc dbmlsync Option [page 130]
ContinueDownload (cd) Extended Option [page 169]
ContinueDownload Synchronization Profile Option [page 211]
sp_hook_dbmlsync_end [page 265]
-kpd dbmlsync Option [page 137]

1.6.5.14 KillConnections Synchronization Profile Option

Drops conflicting locks to the remote database.

Syntax

d= { *ON* | *OFF* }

KillConnections= { *ON* | *OFF* }

Remarks

In cases where dbmlsync must obtain locks on the tables being synchronized, if another connection has a lock on one of these tables, the synchronization may fail or be delayed. Specifying this option forces SQL Anywhere to drop any other connections to the remote database that hold conflicting locks so that synchronization can proceed immediately.

Example

The following example shows how to use the KillConnections synchronization profile option:

```
CREATE SYNCHRONIZATION PROFILE myprofile 'KillConnections=on'
```

Related Information

Concurrency During Synchronization [page 108]
-d dbmlsync Option [page 129]

1.6.5.15 LogRenameSize Synchronization Profile Option

Renames and restarts the transaction log.

Syntax

```
x=size[ K | M ]
```

```
LogRenameSize=size[ K | M ]
```

Remarks

When this option is set, the transaction log is renamed and restarted during synchronization if it is larger than the specified size in bytes. Use the suffix K or M to specify units of kilobytes or megabytes, respectively.

Set the size to 0 to rename the transaction log regardless of its size.

Example

The following example shows how to use the LogRenameSize synchronization profile option:

```
CREATE SYNCHRONIZATION PROFILE myprofile 'LogRenameSize=512k'
```

Related Information

-x dbmlsync Option [page 159]

1.6.5.16 MobiLinkPwd Synchronization Profile Option

Supplies the password of the MobiLink user.

Syntax

```
mp=password
```

```
MobiLinkPwd=password
```

Example

The following example shows how to use the MobiLinkPwd synchronization profile option:

```
CREATE SYNCHRONIZATION PROFILE myprofile 'MobiLinkPwd=mypassword'
```

Related Information

-mp dbmlsync Option [page 139]
MobiLinkPwd (mp) Extended Option [page 181]
NewMobiLinkPwd (mn) Extended Option [page 182]
-mn dbmlsync Option [page 138]

1.6.5.17 MLUser Synchronization Profile Option (Deprecated)

Specifies the MobiLink user name.

Syntax

```
u=username
```

```
MLUser=username
```

Remarks

This option is deprecated. Use the Subscription synchronization profile option instead.

Example

The following example shows how to use the MLUser synchronization profile option:

```
CREATE SYNCHRONIZATION PROFILE myprofile 'MLUser=my_user_name'
```

Related Information

MobiLink Users in a Synchronization System [page 9]

Subscription Synchronization Profile Option [page 226]
-u dbmlsync Option (Deprecated) [page 153]

1.6.5.18 NewMobiLinkPwd Synchronization Profile Option

Supplies a new password for the MobiLink user. Use this option when you want to change the existing password.

Syntax

mn=`new_password`

NewMobiLinkPwd=`new_password`

Example

The following example shows how to use the NewMobiLinkPwd synchronization profile option:

```
CREATE SYNCHRONIZATION PROFILE myprofile 'NewMobiLinkPwd=new_password'
```

Related Information

MobiLink Users in a Synchronization System [page 9]
-mp dbmlsync Option [page 139]
-mn dbmlsync Option [page 138]

1.6.5.19 Ping Synchronization Profile Option

Pings a MobiLink server.

Syntax

pi= { *ON* | *OFF* }

Ping= { *ON* | *OFF* }

Remarks

When you set the ping synchronization profile option to on, dbmlsync connects to the remote database, retrieves information required to connect to the MobiLink server, connects to the server, and authenticates the specified MobiLink user.

When the MobiLink server receives a ping, it connects to the consolidated database, authenticates the user, and then sends the authenticating user status and value back to the client. If the MobiLink user name cannot be found in the ml_user system table and the MobiLink server is running with the command line option -zu+, the MobiLink server adds the user to the ml_user MobiLink system table.

To adequately test your connection, you should use the ping synchronization profile option with all the synchronization options you want to use to synchronize with dbmlsync. When the ping synchronization profile option is included, dbmlsync does not perform a synchronization.

If the ping succeeds, the MobiLink server issues an information message. If the ping does not succeed, it issues an error message.

Example

The following example shows how to use the Ping synchronization profile option:

```
CREATE SYNCHRONIZATION PROFILE myprofile 'Ping=on'
```

Related Information

-pi dbmlsync Option [page 144]

1.6.5.20 Publication Synchronization Profile Option (Deprecated)

Specifies the publications(s) to synchronize.

Syntax

```
n=pubname, ...
```

```
Publication=pubname, ...
```

Remarks

The publication synchronization profile option can be specified only once in a synchronization profile.

> **Note**
>
> This option has been deprecated. Use either the Subscription synchronization profile option or the -s dbmlsync option instead.
>
> To use the dbmlsync -s option you need to determine the subscription name for the subscription you want to synchronize. You can determine the subscription name using the following query:
>
> ```
> SELECT subscription_name
> FROM syssync JOIN sys.syspublication
> WHERE site_name = <ml_user> AND publication_name = <pub_name>;
> ```
>
> Replace <ml_user> with the MobiLink user you are synchronizing. This is the value specified by the -u option on the dbmlsync command line.
>
> Replace <pub_name> with the name of the publication being synchronized. This is the value specified with the -n option on the dbmlsync command line.

Example

```
CREATE SYNCHRONIZATION PROFILE myprofile 'Publication=overnight'
```

Related Information

Subscription Synchronization Profile Option [page 226]
-s dbmlsync Option [page 148]
-u dbmlsync Option (Deprecated) [page 153]
-n dbmlsync Option (Deprecated) [page 140]

1.6.5.21 RemoteProgressGreater Synchronization Profile Option

Specifies that the remote offset should be used if it is greater than the consolidated offset. This is equivalent to the -ra option.

> Syntax
>
> *ra=* { *ON* | *OFF* }

RemoteProgressGreater={ *ON* | *OFF* }

Remarks

This option should be used only in very rare cases. If you use this option, the upload is retried starting from the offset obtained from the remote database if the remote offset is greater than the offset obtained from the consolidated database. If you use this option and the offset in the remote database is not greater than the offset from the consolidated database, an error is reported and the synchronization is aborted.

This option should be used with care. If the offset mismatch is the result of a restore of the consolidated database, changes that happened in the remote database in the gap between the two offsets are lost. This option may be useful when the consolidated database has been restored from backup and the remote database transaction log has been truncated at the same point as the remote offset. In this case, all data that was uploaded from the remote database is lost from the point of the consolidated offset to the point of the remote offset.

Example

```
CREATE SYNCHRONIZATION PROFILE myprofile 'RemoteProgressGreater=on'
```

Related Information

-r dbmlsync Option [page 147]

1.6.5.22 RemoteProgressLess Synchronization Profile Option

Specifies that the remote offset should be used if it is less than the consolidated offset (such as when the remote database has been restored from backup). This is equivalent to the -rb option.

Syntax

rb={ *ON* | *OFF* }

RemoteProgressLess={ *ON* | *OFF* }

Remarks

If the remote database is restored from backup, the default behavior may cause data to be lost. When you use this option, the upload continues from the offset recorded in the remote database if the offset recorded in the remote database is less than that obtained from the consolidated database. If you use this option and the offset in the remote database is not less than the offset from the consolidated database, an error is reported and the synchronization is aborted.

This option may result in some data being uploaded that has already been uploaded. This can result in conflicts in the consolidated database and should be handled with appropriate conflict resolution scripts.

Example

```
CREATE SYNCHRONIZATION PROFILE myprofile 'RemoteProgressLess=on'
```

Related Information

-r dbmlsync Option [page 147]

1.6.5.23 Subscription Synchronization Profile Option

Specifies the subscription(s) to synchronize.

Syntax

```
S=subname, ...
```

```
Subscription=subname, ...
```

Remarks

Subscription can only be specified once in a synchronization profile but the command line option can be specified multiple times.

Example

```
CREATE SYNCHRONIZATION PROFILE myprofile 'Subscription=mySubscription'
```

Related Information

-s dbmlsync Option [page 148]

1.6.5.24 TransactionalUpload Synchronization Profile Option

Boolean. Specifies that each transaction on the remote database should be uploaded as a separate transaction within one synchronization.

Syntax

tu= { *ON* | *OFF* }

TransactionalUpload= { *ON* | *OFF* }

Remarks

When you use the TransactionalUpload synchronization profile option, you create a transactional upload: 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.

Example

```
CREATE SYNCHRONIZATION PROFILE myprofile 'TransactionalUpload=on'
```

Related Information

-tu dbmlsync Option [page 152]

1.6.5.25 UpdateGenNum Synchronization Profile Option

When creating a download file, this option creates a file that can be used with remote databases that have not yet synchronize. Otherwise, you must perform a synchronization before you apply a download file.

Syntax

bg= { *ON* | *OFF* }

UpdateGenNum= { *ON* | *OFF* }

Remarks

This option causes the download file to update the generation numbers on the remote database.

Download files built with this option should be snapshot downloads. Timestamp-based downloads do not work with remote databases that have not synchronized because the last download timestamp on a new remote database is by default January 1, 1900, which is earlier than the last download timestamp in the download file. For timestamp-based file-based downloads to work, the last download timestamp in the download file must be the same or earlier than on the remote.

Do not apply UpdateGenNum download files to remote databases that have already synchronized if your system depends on functionality provided by generation numbers as this option circumvents that functionality.

Example

```
CREATE SYNCHRONIZATION PROFILE myprofile 'UpdateGenNum=on'
```

Related Information

MobiLink Generation Numbers
Application of the Download File
-bg dbmlsync Option [page 125]

1.6.5.26 UploadOnly Synchronization Profile Option

Specifies that synchronization only includes an upload, and that no download should occur.

Syntax

uo= { *ON* | *OFF* }

UploadOnly= { *ON* | *OFF* }

Remarks

During upload-only synchronization, dbmlsync prepares and sends an upload to MobiLink exactly as it would in a normal full synchronization. However, instead of sending a download back down, MobiLink sends only an acknowledgement indicating if the upload was successfully committed.

Refer to the documentation on required synchronization scripts for a list of the scripts that must be defined for upload-only synchronization.

Example

```
CREATE SYNCHRONIZATION PROFILE myprofile 'UploadOnly=on'
```

Related Information

Scripts Required for Synchronization
-uo dbmlsync Option [page 155]
UploadOnly (uo) Extended Option [page 194]

1.6.5.27 UploadRowCnt Synchronization Profile Option

Integer. Specifies an estimate of the number of rows to be uploaded in a synchronization.

Syntax

urc=`rowcount`

UploadRowCnt=`rowcount`

Remarks

To improve performance, you can specify an estimate of the number of rows to upload in a synchronization. This setting is especially useful when you are uploading a large number of rows. A higher estimate results in faster uploads but more memory usage.

Synchronization proceeds correctly regardless of the specified estimate.

Example

```
CREATE SYNCHRONIZATION PROFILE myprofile 'UploadRowCnt=100'
```

Related Information

-urc dbmlsync Option [page 156]

1.6.5.28 Verbosity Synchronization Profile Option

Controls dbmlsync verbosity.

Syntax

v= { *BASIC* | *HIGH* | *CONNECT_STR* | *ROW_CNT* | *OPTIONS* | *ML_PASSWORD* | *ROW_DATA* | *HOOK* }, ...

Verbosity= { *BASIC* | *HIGH* | *CONNECT_STR* | *ROW_CNT* | *OPTIONS* | *ML_PASSWORD* | *ROW_DATA* | *HOOK* }, ...

Allowed values

The value must be a comma-separated list of one or more of the following options, each of which enables a different type of verbosity:

BASIC

Generate limited verbosity.

HIGH

Generate the maximum possible level of verbosity.

CONNECT_STR

Expose the connection strong in the log.

ROW_CNT

Log the number of rows that were uploaded and downloaded.

OPTIONS

Log the options used to specify synchronization.

ML_PASSWORD

Expose the MobiLink password in the log.

ROW_DATA

Log rows that were uploaded and downloaded.

HOOK

Log messages related to hook scripts.

Remarks

If you specify a verbosity extended option and a verbosity synchronization profile option and there are conflicts, the verbosity synchronization profile option overrides the verbosity extended option.

If you specify both -v and a verbosity extended option and there are conflicts, the -v option overrides the extended option. If there is no conflict, the verbosity logging options are additive - all options that you specify are used. If you use the -v command line option, the verbosity options take effect immediately. If you use the extended option, the verbosity options do not take effect until the first synchronization begins, so startup information is not logged. After the first synchronization, the behavior should be the same regardless of how the option was specified.

Example

```
CREATE SYNCHRONIZATION PROFILE myprofile 'Verbosity=OPTIONS, ML_PASSWORD'
```

Related Information

-v dbmlsync Option [page 157]

1.6.6 Event Hooks for SQL Anywhere Clients

The SQL Anywhere synchronization client, dbmlsync, provides an optional set of event hooks that you can use to customize the synchronization process. When a hook is implemented, it is called at a specific point in the synchronization process.

You implement an event hook by creating a SQL stored procedure with a specific name. Most event-hook stored procedures are executed on the same connection as the synchronization itself.

You can use event hooks to log and handle synchronization events. For example, you can schedule synchronizations based on logical events, retry connection failures, or handle errors and referential integrity violations.

In addition, you can use event hooks to synchronize subsets of data that cannot be easily defined in a publication. For example, you can synchronize data in a temporary table by writing one event hook procedure to copy data from the temporary table to a permanent table before the synchronization and another to copy the data back afterward.

> ⚠ Caution
>
> The integrity of the synchronization process relies on a sequence of built-in transactions. You must not perform an implicit or explicit commit or rollback within your event-hook procedures.
>
> If you change any connection setting in a hook you must restore the setting to its previous value before the hook ends. Failing to restore the setting may produce unexpected results.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

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.

In this section:

sp_hook_dbmlsync_abort [page 240]
    Use this stored procedure to cancel the synchronization process.

sp_hook_dbmlsync_all_error [page 243]
    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.

sp_hook_dbmlsync_begin [page 246]
    Use this stored procedure to add custom actions at the beginning of the synchronization process.

sp_hook_dbmlsync_communication_error [page 247]
    Use this stored procedure to process communications errors.

sp_hook_dbmlsync_delay [page 250]
    Use this stored procedure to control when synchronization takes place.

sp_hook_dbmlsync_download_begin [page 253]
    Use this stored procedure to add custom actions at the beginning of the download stage of the synchronization process.

sp_hook_dbmlsync_download_end [page 255]
    Use this stored procedure to add custom actions at the end of the download stage of the synchronization process.

sp_hook_dbmlsync_download_log_ri_violation [page 257]
    Logs referential integrity violations in the download process.

sp_hook_dbmlsync_download_ri_violation [page 259]
    Allows you to resolve referential integrity violations in the download process.

sp_hook_dbmlsync_download_table_begin [page 261]
    Use this stored procedure to add custom actions immediately before each table is downloaded.

sp_hook_dbmlsync_download_table_end [page 263]
    Use this stored procedure to add custom actions immediately after each table is downloaded.

sp_hook_dbmlsync_end [page 265]
    Use this stored procedure to add custom actions immediately before synchronization is complete.

sp_hook_dbmlsync_log_rescan [page 268]
    Use this stored procedure to programmatically decide when a rescan is required.

sp_hook_dbmlsync_logscan_begin [page 270]
    Use this stored procedure to add custom actions immediately before the transaction log is scanned for upload.

sp_hook_dbmlsync_logscan_end [page 272]
    Use this stored procedure to add custom actions immediately after the transaction log is scanned.

sp_hook_dbmlsync_misc_error [page 274]
    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.

sp_hook_dbmlsync_ml_connect_failed [page 277]
    Use this stored procedure to retry failed attempts to connect to the MobiLink server using a different communication type or address.

sp_hook_dbmlsync_process_exit_code [page 280]

Use this stored procedure to manage exit codes.

sp_hook_dbmlsync_schema_upgrade [page 283]
  Use this stored procedure to run a SQL script that revises your schema.

sp_hook_dbmlsync_set_extended_options [page 285]
  Use this stored procedure to programmatically customize the behavior of an upcoming synchronization by specifying extended options to be applied to that synchronization.

sp_hook_dbmlsync_set_ml_connect_info [page 287]
  Use this stored procedure to set the network protocol and network protocol option used to connect to the MobiLink server.

sp_hook_dbmlsync_set_upload_end_progress [page 289]
  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.

sp_hook_dbmlsync_sql_error [page 291]
  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.

sp_hook_dbmlsync_upload_begin [page 293]
  Use this stored procedure to add custom actions immediately before the transmission of the upload.

sp_hook_dbmlsync_upload_end [page 295]
  Use this stored procedure to add custom actions after dbmlsync has verified receipt of the upload by the MobiLink server.

sp_hook_dbmlsync_validate_download_file [page 298]
  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.

Related Information

dbmlsync Synchronization Customization [page 112]

1.6.6.1 Synchronization Event Hook Sequence

The following pseudo-code shows the available events and the point at which each is called during the synchronization process. For example, sp_hook_dbmlsync_abort is the first event hook to be invoked.

```
sp_hook_dbmlsync_abort //not called when Dbmlsync API or the SQL SYNCHRONIZE
STATEMENT is used
sp_hook_dbmlsync_set_extended_options
loop until return codes direct otherwise (
   sp_hook_dbmlsync_abort
   sp_hook_dbmlsync_delay
)
sp_hook_dbmlsync_abort
// start synchronization
sp_hook_dbmlsync_begin
// upload events
```

```
for each upload segment
// a normal synchronization has one upload segment
// a transactional upload has one segment per transaction
// an incremental upload has one segment per upload piece
 sp_hook_dbmlsync_logscan_begin  //not called for scripted upload
 sp_hook_dbmlsync_logscan_end  //not called for scripted upload
 sp_hook_dbmlsync_set_ml_connect_info //only called during first upload
 sp_hook_dbmlsync_upload_begin
 sp_hook_dbmlsync_set_upload_end_progress  //only called for scripted upload
 sp_hook_dbmlsync_upload_end
next upload segment
// download events
sp_hook_dbmlsync_validate_download_file (only called
   when -ba option is used)
sp_hook_dbmlsync_download_begin
for each table
   sp_hook_dbmlsync_download_table_begin
   sp_hook_dbmlsync_download_table_end
next table
sp_hook_dbmlsync_download_end
sp_hook_dbmlsync_schema_upgrade
// end synchronization
sp_hook_dbmlsync_end
sp_hook_dbmlsync_process_exit_code
sp_hook_dbmlsync_log_rescan
```

Event hooks

Each hook is provided with parameter values that you can use when you implement the procedure. Sometimes, you can modify the value to return a new value; others are read-only. These parameters are not stored procedure arguments. No arguments are passed to any of the event-hook stored procedures. Instead, arguments are exchanged by reading and modifying rows in the #hook_dict table.

For example, the sp_hook_dbmlsync_begin procedure has a MobiLink user parameter, which is the MobiLink user being synchronized. You can retrieve this value from the #hook_dict table.

Although the sequence has similarities to the event sequence at the MobiLink server, there is little overlap in the kind of logic you would want to add to the consolidated and remote databases. The two interfaces are therefore separate and distinct.

If a *_begin hook executes successfully, the corresponding *_end hook is called regardless of any error that occurred after the *_begin hook. If the *_begin hook is not defined, but you have defined an *_end hook, then the *_end hook is called unless an error occurs before the point in time where the *_begin hook would normally be called.

If the hooks change data in your database, all changes up to and including sp_hook_dbmlsync_logscan_begin are synchronized in the current synchronization session; after that point, changes are synchronized in the next session.

1.6.6.2 Event-hook Procedures

There are some considerations to keep in mind for designing and using event-hook procedures.

- Do not perform any COMMIT or ROLLBACK operations in event-hook procedures. The procedures are executed on the same connection as the synchronization, and a COMMIT or ROLLBACK may interfere with synchronization.
- If you change any connection setting in a hook you must restore the setting to its previous value before the hook ends. Failing to restore the setting may produce unexpected results.
- Dbmlsync 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, or a group of which that user is a member.
- Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:
  - Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
  - Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

In this section:

1.6.6.2.1 #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. Sometimes, you can modify the value to return a new value; others are read-only. Each row in the table contains the value for one parameter.

For example, suppose two subscriptions are defined as follows:

```
CREATE SYNCHRONIZATION SUBSCRIPTION sub1
```

```
TO pub1
FOR  MyUser;
SCRIPT VERSION 'v1'
```

```
CREATE SYNCHRONIZATION SUBSCRIPTION sub2
TO pub2
FOR  MyUser;
SCRIPT VERSION 'v1'
```

When the sp_hook_dbmlsync_begin hook is called for the following dbmlsync command line

```
dbmlsync -c 'DSN=MyDsn' -s sub1,sub2
```

the #hook_dict table contains the following rows:

Name	Value
subscription_0	sub1
subscription_1	sub2
publication_0	pub1
publication_1	pub2
MobiLink user	MyUser
Script version	v1

i Note

The publication_`n` rows are deprecated and may be removed in a future release.

A hook can retrieve values from the #hook_dict table and use them to customize behavior. For example, to retrieve the MobiLink user you would use a SELECT statement like this:

```
SELECT value
FROM #hook_dict
WHERE name = 'MobiLink user'
```

In/out parameters can be updated by your hook to modify the behavior of dbmlsync. For example, in the sp_hook_dbmlsync_abort hook you could instruct dbmlsync to abort synchronization by updating the abort synchronization row of the table using a statement like this:

```
UPDATE #hook_dict
SET value='true'
WHERE name='abort synchronization'
```

The description of each hook lists the rows in the #hook_dict table.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

Example

The following sample sp_hook_dbmlsync_delay procedure illustrates the use of in/out parameters in the #hook_dict table. The procedure allows synchronization only outside a scheduled down time of the MobiLink system between 18:00 and 19:00.

```
CREATE PROCEDURE sp_hook_dbmlsync_delay()
BEGIN
   DECLARE delay_val integer;
 SET delay_val=DATEDIFF(
   second, CURRENT TIME, '19:00');
 IF (delay_val>0 AND
     delay_val<3600)
 THEN
 UPDATE #hook_dict SET value=delay_val
   WHERE name='delay duration';
 END IF;
END
```

The following procedure is executed in the remote database at the beginning of synchronization. It retrieves the current MobiLink user name (one of the parameters available for the sp_hook_dbmlsync_begin event), and displays it on the SQL Anywhere messages window.

```
CREATE PROCEDURE sp_hook_dbmlsync_begin()
BEGIN
   DECLARE MLuser VARCHAR(150);
   SELECT '>>>MLuser = ' || value INTO MLuser
      FROM #hook_dict
      WHERE name ='MobiLink user';
   MESSAGE MLuser TYPE INFO TO CONSOLE;
END
```

1.6.6.2.2 Connections for Event-hook Procedures

Each event-hook procedure is executed on the same connection as the synchronization itself. However, there are a few exceptions.

The exceptions are as follows:

- sp_hook_dbmlsync_all_error
- sp_hook_dbmlsync_communication_error

- sp_hook_dbmlsync_download_log_ri_violation
- sp_hook_dbmlsync_misc_error
- sp_hook_dbmlsync_sql_error

These procedures are called before a synchronization fails. On failure, synchronization actions are rolled back. By executing on a separate connection, you can use these procedures to log information about the failure, without the logging actions being rolled back along with the synchronization actions.

1.6.6.2.3 Error and Warning Handling in Event Hook Procedures

You can create event hook stored procedures to handle synchronization errors, MobiLink connection failures, and referential integrity violations. 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
- sp_hook_dbmlsync_download_ri_violation

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.

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.

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

Ignoring errors

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

Related Information

sp_hook_dbmlsync_download_log_ri_violation [page 257]
sp_hook_dbmlsync_download_ri_violation [page 259]
sp_hook_dbmlsync_ml_connect_failed [page 277]
sp_hook_dbmlsync_communication_error [page 247]
sp_hook_dbmlsync_sql_error [page 291]
sp_hook_dbmlsync_misc_error [page 274]
sp_hook_dbmlsync_all_error [page 243]
sp_hook_dbmlsync_end [page 265]
IgnoreHookErrors (eh) Extended Option [page 177]

1.6.6.3 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 synchronization terminates immediately after the event.
*publication*_n (in)	publication	Deprecated. Use subscription_n instead. The publications being synchronized, where n is an integer. There is one publication_n entry for each publication being synchronized. The numbering of n starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.

Name	Value	Description
exit code (in|out)	number	When abort synchronization is set to TRUE, you can use this value to set the exit code for the aborted synchronization. 0 indicates a successful synchronization. Any other number indicates that the synchronization failed.
script version (in|out)	script version name	The MobiLink script version to be used for the synchronization.
*subscription_*`n` (in)	subscription name(s)	The names of subscriptions being synchronized where `n` is an integer. There is one subscription_n entry for each subscription being synchronized. The numbering of `n` starts at zero.

Remarks

If a procedure of this name exists, it is called at dbmlsync startup, and then again after each synchronization delay that is caused by the sp_hook_dbmlsync_delay hook.

When dbmlsync is run from the command line, setting the abort synchronization to true causes all remaining synchronizations (including scheduled synchronizations) to be canceled. When the dbmlsync API or the SQL SYNCHRONIZE statement is used, setting abort synchronization to true only causes the current synchronization to be aborted.

Actions of this procedure are committed immediately after execution.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

Example

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;
```

Related Information

Synchronization Event Hook Sequence [page 234]

sp_hook_dbmlsync_process_exit_code [page 280]

1.6.6.4 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*	Deprecated. Use subscription_n instead. The publications being synchronized, where n is an integer. There is one publication_n entry for each publication being synchronized. The numbering of n starts at zero.
MobiLink user (in)	*MobiLink user name*	The MobiLink user for which you are synchronizing.
script version (in)	*script version name*	The MobiLink script version that is used for the synchronization.
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 a hook changes the value of the row, the new value is used in the next hook call.
subscription_n (in)	*subscription name(s)*	The names of subscriptions being synchronized where n is an integer. This is one subscription_n entry for each subscription being synchronized. The numbering of n starts at zero.

Remarks

Each time a dbmlsync error message is generated, the following hooks are called:

- First, depending on the type of error, one of the following hooks is called: sp_hook_dbmlsync_communication_error, sp_hook_dbmlsync_misc_error, or sp_hook_dbmlsync_sql_error. These hooks contain information specific to the type of error; for example, sqlcode and sqlstate are provided for SQL errors.
- Next, the sp_hook_dbmlsync_all_error is called. This hook is useful for logging all errors that occurred.

If an error occurs during startup before a synchronization has been initiated, the #hook_dict entries for MobiLink user and Script version are set to an empty string, and no publication_`n` or subscription_`n` rows are set in the #hook_dict table.

The error hook user state row provides a useful mechanism for you to pass information about the nature of the error to the sp_hook_dbmlsync_end hook, where you might use that information to decide whether to retry the synchronization.

This procedure executes on a separate connection to ensure that operations it performs are not lost if a rollback is performed on the synchronization connection. If dbmlsync cannot establish a separate connection, the procedure is not called.

Since this hook executes on a separate connection you should use care when accessing tables that are being synchronized in your hook procedure because dbmlsync may have locks on these tables. These locks could cause operations in your hook to fail or to wait indefinitely.

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

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

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;
```

Related Information

Error and Warning Handling in Event Hook Procedures [page 239]

sp_hook_dbmlsync_communication_error [page 247]

sp_hook_dbmlsync_misc_error [page 274]

sp_hook_dbmlsync_sql_error [page 291]

sp_hook_dbmlsync_end [page 265]

1.6.6.5 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`	Deprecated. Use *subscription*_`n` instead. The publications being synchronized, where `n` is an integer. There is one *publication*_`n` entry for each publication being synchronized. The numbering of `n` starts at zero.
MobiLink user (in)	`MobiLink user name`	The MobiLink user for which you are synchronizing.
script version (in)	`script version name`	The MobiLink script version to be used for the synchronization.
*subscription*_`n` (in)	`subscription name(s)`	The names of subscriptions being synchronized where `n` is an integer. This is one *subscription*_`n` entry for each subscription being synchronized. The numbering of `n` starts at zero.

Remarks

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

Actions of this procedure are committed immediately after execution.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

Example

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
   "event_id"           integer NOT NULL DEFAULT AUTOINCREMENT ,
   "event_time"         timestamp NULL,
   "event_name"         varchar(128) NOT NULL ,
   "subs"               varchar(1024) NULL ,
    PRIMARY KEY ("event_id")
)
```

The following logs the beginning of each synchronization in the table.

```
CREATE PROCEDURE sp_hook_dbmlsync_begin ()
BEGIN

    DECLARE subs_list VARCHAR(1024);
-- build a list of subscriptions being synchronized
    SELECT LIST(value) INTO subs_list
     FROM #hook_dict
     WHERE name LIKE 'subscription_%';
-- log the event
    INSERT INTO SyncLog(event_time, event_name, subs)
    VALUES( CURRENT TIMESTAMP, 'sp_hook_dbmlsync_begin', subs_list );
END
```

Related Information

Synchronization Event Hook Sequence [page 234]

1.6.6.6 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	Deprecated. Use subscription_n instead. The publications being synchronized, where n is an integer. There is one publication_n entry for each publication being synchronized. The numbering of n starts at zero.

Name	Value	Description
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version 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 a hook changes the value of the row, the new value is used in the next hook call.
stream error code (in)	integer	The error reported by the stream.
system error code (in)	integer	A system-specific error code.
*subscription_*`n` (in)	subscription name(s)	The names of subscriptions being synchronized where `n` is an integer. This is one subscription_`n` entry for each subscription being synchronized. The numbering of `n` starts at zero.

Remarks

If an error occurs during startup before a synchronization has been initiated, the #hook_dict entries for MobiLink user and Script version are set to an empty string, and no publication_`n` or subscription_`n`rows are set in the #hook_dict table.

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.

The error hook user state row provides a useful mechanism for you to pass information about the nature of the error to the sp_hook_dbmlsync_end hook, where you might use that information to decide whether to retry the synchronization.

This procedure executes on a separate connection to ensure that operations it performs are not lost if a rollback is performed on the synchronization connection. If dbmlsync cannot establish a separate connection, the procedure is not called.

Since this hook executes on a separate connection you should use care when accessing tables that are being synchronized in your hook procedure because dbmlsync may have locks on these tables. These locks could cause operations in your hook to fail or to wait indefinitely.

Actions of this procedure are committed immediately after execution.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

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
```

Related Information

MobiLink Communication Error Messages
Error and Warning Handling in Event Hook Procedures [page 239]
sp_hook_dbmlsync_all_error [page 243]
sp_hook_dbmlsync_misc_error [page 274]
sp_hook_dbmlsync_sql_error [page 291]

1.6.6.7 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 immediately. A non-zero delay duration value specifies the number of seconds before the delay hook is called again.
maximum accumulated delay (in|out)	number of seconds	The maximum accumulated delay specifies the maximum number of seconds delay before each synchronization. Dbmlsync keeps track of the total delay created by all calls to the delay hook since the last synchronization. If no synchronization has occurred since dbmlsync started running, the total delay is calculated from the time dbmlsync started up. When the total delay exceeds the value of maximum accumulated delay, synchronization begins without any further calls to the delay hook.

Name	Value	Description
*publication*_n (in)	publication	Deprecated. Use subscription_n instead. The publications being synchronized, where n is an integer. There is one publication_n entry for each publication being synchronized. The numbering of n starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
*subscription*_n (in)	subscription name(s)	The names of subscriptions being synchronized where n is an integer. This is one subscription_n entry for each subscription being synchronized. The numbering of n starts at zero.

Remarks

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

This hook is not called when synchronization is initiated using the Dbmlsync API or the SQL SYNCHRONIZE statement.

Actions of this procedure are committed immediately after execution.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

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;
```

This example assumes that you have used LockTables extended option to ensure that the tables are locked during synchronization. If the tables are not locked, it is possible for a high priority row to be inserted after the upload is built but before the sp_hook_dbmlsync_end hook is executed. If that happened the row's priority would be changed to "high-processed" even though it was never uploaded.

Related Information

Synchronization Event Hook Sequence [page 234]

Synchronization Initiation with Event Hooks [page 112]

sp_hook_dbmlsync_download_end [page 255]

sp_hook_dbmlsync_end [page 265]

1.6.6.8 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`	Deprecated. Use *subscription*_`n` instead. The publications being synchronized, where `n` is an integer. There is one *publication*_`n` entry for each publication being synchronized. The numbering of `n` starts at zero.
MobiLink user (in)	`MobiLink user name`	The MobiLink user for which you are synchronizing.
script version (in)	`script version name`	The MobiLink script version to be used for the synchronization.
*subscription*_`n` (in)	`subscription name(s)`	The names of subscriptions being synchronized where `n` is an integer. This is one *subscription*_`n` entry for each subscription being synchronized. The numbering of `n` starts at zero.

Remarks

If a procedure of this name exists, it is called at the beginning of the download stage of the synchronization process.

Actions of this procedure are committed or rolled back when the download is committed or rolled back.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

Example

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
   "event_id"           integer NOT NULL DEFAULT AUTOINCREMENT ,
   "event_time"         timestamp NULL,
   "event_name"         varchar(128) NOT NULL ,
   "subs"               varchar(1024) NULL ,
    PRIMARY KEY ("event_id")
)
```

The following logs the beginning of the download for each synchronization.

```
CREATE PROCEDURE sp_hook_dbmlsync_download_begin ()
BEGIN

    DECLARE subs_list VARCHAR(1024);
-- build a list of subscriptions being synchronized
    SELECT LIST(value) INTO subs_list
     FROM #hook_dict
     WHERE name LIKE 'subscription_%';
-- log the event
    INSERT INTO SyncLog(event_time, event_name, subs)
    VALUES( CURRENT TIMESTAMP, 'sp_hook_dbmlsync_download_begin', subs_list );
END
```

Related Information

Synchronization Event Hook Sequence [page 234]

1.6.6.9 sp_hook_dbmlsync_download_end

Use this stored procedure to add custom actions at the end of the download stage of the synchronization process.

Rows in #hook_dict table

Name	Value	Description
*publication*_`n` (in)	`publication`	Deprecated. Use *subscription*_`n` instead. The publications being synchronized, where `n` is an integer. There is one *publication*_`n` entry for each publication being synchronized. The numbering of `n` starts at zero.
MobiLink user (in)	`MobiLink user name`	The MobiLink user for which you are synchronizing.
script version (in)	`script version name`	The MobiLink script version to be used for the synchronization.
*subscription*_n (in)	`subscription name(s)`	The names of subscriptions being synchronized where `n` is an integer. This is one *subscription*_`n` entry for each subscription being synchronized. The numbering of `n` starts at zero.

Remarks

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

Actions of this procedure are committed or rolled back when the download is committed or rolled back.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

Example

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
   "event_id"           integer NOT NULL DEFAULT autoincrement ,
   "event_time"         timestamp NULL,
   "event_name"         varchar(128) NOT NULL ,
   "subs"               varchar(1024) NULL ,
    PRIMARY KEY ("event_id")
)
```

The following logs the end of the download for each synchronization.

```
CREATE PROCEDURE sp_hook_dbmlsync_download_end ()
BEGIN

    DECLARE subs_list VARCHAR(1024);
-- build a list of subscriptions being synchronized
    SELECT LIST(value) INTO subs_list
     FROM #hook_dict
     WHERE name LIKE 'subscription_%';
-- log the event
    INSERT INTO SyncLog(event_time, event_name, subs)
    VALUES( CURRENT TIMESTAMP, 'sp_hook_dbmlsync_download_end', subs_list );
END
```

Related Information

Synchronization Event Hook Sequence [page 234]

Synchronization Initiation with Event Hooks [page 112]

sp_hook_dbmlsync_delay [page 250]

1.6.6.10 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`	Deprecated. Use *subscription*_`n` instead. The publications being synchronized, where `n` is an integer. There is one *publication*_`n` entry for each publication being synchronized. The numbering of `n` starts at zero.
MobiLink user (in)	`MobiLink user name`	The MobiLink user for which you are synchronizing.
foreign key table (in)	`table name`	The table containing the foreign key column for which the hook is being called.
primary key table (in)	`table name`	The table referenced by the foreign key for which the hook is being called.
role name (in)	`role name`	The role name of the foreign key for which the hook is being called.
script version (in)	`script version name`	The MobiLink script version to be used for the synchronization.
*subscription*_`n` (in)	`subscription name(s)`	The names of subscriptions being synchronized where `n` is an integer. This is one *subscription*_`n` entry for each subscription being synchronized. The numbering of `n` starts at zero.

Remarks

A download RI violation occurs when rows in the download violate foreign key relationships on the remote database. This hook allows you to log RI violations as they occur so that you can investigate their cause later.

After the download is complete, but before it is committed, 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_violation (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.

Since this hook executes on a separate connection you should use care when accessing tables that are being synchronized in your hook procedure because dbmlsync may have locks on these tables. These locks could cause operations in your hook to fail or to wait indefinitely.

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.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

Example

Assume you use the following table to log referential integrity violations.

```
CREATE TABLE DBA.LogRIViolationTable
(
    entry_time  TIMESTAMP,
    pk_table  VARCHAR( 255 ),
    fk_table  VARCHAR( 255 ),
    role_name  VARCHAR( 255 )
);
```

The following example logs the foreign key table name, primary key table name, and role name when a referential integrity violation is detected on the remote database. The information is stored in LogRIViolationTable on the remote database.

```
CREATE PROCEDURE sp_hook_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;
```

Related Information

Synchronization Event Hook Sequence [page 234]
sp_hook_dbmlsync_download_ri_violation [page 259]

1.6.6.11 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`	Deprecated. Use *subscription*_n instead. The publications being synchronized, where n is an integer. There is one *publication*_n entry for each publication being synchronized. The numbering of n starts at zero.
MobiLink user (in)	`MobiLink user name`	The MobiLink user for which you are synchronizing.
foreign key table (in)	`table name`	The table containing the foreign key column for which the hook is being called.
primary key table (in)	`table name`	The table referenced by the foreign key for which the hook is being called.
role name (in)	`role name`	The role name of the foreign key for which the hook is being called.
script version (in)	`script version name`	The MobiLink script version to be used for the synchronization.
*subscription*_n (in)	`subscription name(s)`	The names of subscriptions being synchronized where n is an integer. This is one *subscription*_n entry for each subscription being synchronized. The numbering of n starts at zero.

Remarks

A download RI violation occurs when rows in the download violate foreign key relationships on the remote database. This hook allows you to attempt to resolve RI violations before 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_violation (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.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

Example

This example uses the Department and Employee tables shown below:

```
CREATE TABLE Department(
 "department_id"  INTEGER primary key
);
CREATE TABLE Employee(
  "employee_id"      INTEGER PRIMARY KEY,
  "department_id" INTEGER,
  FOREIGN KEY EMPLOYEE_FK1 (department_id) REFERENCES Department
);
```

The following sp_hook_dbmlsync_download_ri_violation definition cleans up referential integrity violations between the Department and Employee tables. It verifies the role name for the foreign key and inserts missing department_id values into the Department table.

```
CREATE PROCEDURE sp_hook_dbmlsync_download_ri_violation()
BEGIN
```

```
IF EXISTS (SELECT * FROM #hook_dict WHERE name = 'role name'
 AND value = 'EMPLOYEE_FK1') THEN
 -- update the Department table with missing department_id values
 INSERT INTO Department
  SELECT distinct department_id FROM Employee
  WHERE department_id NOT IN (SELECT department_id FROM Department)
END IF;
END;
```

Related Information

sp_hook_dbmlsync_download_log_ri_violation [page 257]

1.6.6.12 sp_hook_dbmlsync_download_table_begin

Use this stored procedure to add custom actions immediately before each table is downloaded.

Rows in #hook_dict table

Name	Value	Description
table name (in)	`table name`	The table to which operations are about to be applied.
*publication*_`n` (in)	`publication`	Deprecated. Use *subscription*_`n` instead. The publications being synchronized, where `n` is an integer. There is one *publication*_`n` entry for each publication being synchronized. The numbering of `n` starts at zero.
MobiLink user (in)	`MobiLink user name`	The MobiLink user for which you are synchronizing.
script version (in)	`script version name`	The MobiLink script version to be used for the synchronization.
*subscription*_`n` (in)	`subscription name(s)`	The names of subscriptions being synchronized where `n` is an integer. This is one *subscription*_`n` entry for each subscription being synchronized. The numbering of `n` starts at zero.

Remarks

If a procedure of this name exists, it is called for each table immediately before downloaded operations are applied to that table. Actions of this procedure are committed or rolled back when the download is committed or rolled back.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

Example

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
   "event_id"            integer NOT NULL DEFAULT autoincrement ,
   "event_time"          timestamp NULL,
   "event_name"          varchar(128) NOT NULL ,
   "subs"                varchar(1024) NULL ,
    PRIMARY KEY ("event_id")
)
```

The following logs the beginning of each table's download for each synchronization.

```
CREATE PROCEDURE sp_hook_dbmlsync_download_table_begin ()
BEGIN

    DECLARE subs_list VARCHAR(1024);
-- build a list of subscriptions being synchronized
    SELECT LIST(value) INTO subs_list
     FROM #hook_dict
     WHERE name LIKE 'subscription_%';
-- log the event
    INSERT INTO SyncLog(event_time, event_name, subs)
    VALUES( CURRENT TIMESTAMP, 'sp_hook_dbmlsync_download_table_begin,
subs_list );
END
```

Related Information

Synchronization Event Hook Sequence [page 234]

1.6.6.13 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`	Deprecated. Use *subscription_*`n` instead. The publications being synchronized, where `n` is an integer. There is one *publication_*`n` entry for each publication being synchronized. The numbering of `n` starts at zero.
MobiLink user (in)	`MobiLink user name`	The MobiLink user for which you are synchronizing.
script version (in)	`script version name`	The MobiLink script version to be used for the synchronization.
*subscription_*`n` (in)	`subscription name(s)`	The names of subscriptions being synchronized where `n` is an integer. This is one *subscription_*`n` entry for each subscription being synchronized. The numbering of `n` starts at zero.

Remarks

If a procedure of this name exists, it is called immediately after all operations in the download for a table have been applied.

Actions of this procedure are committed or rolled back when the download is committed or rolled back.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

Example

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
   "event_id"            integer NOT NULL DEFAULT autoincrement ,
   "event_time"          timestamp NULL,
   "event_name"          varchar(128) NOT NULL ,
   "subs"                varchar(1024) NULL ,
    PRIMARY KEY ("event_id")
)
```

The following logs the end of the download for each table for each synchronization.

```
CREATE PROCEDURE sp_hook_dbmlsync_download_table_end ()
BEGIN

    DECLARE subs_list VARCHAR(1024);
-- build a list of subscriptions being synchronized
    SELECT LIST(value) INTO subs_list
     FROM #hook_dict
     WHERE name LIKE 'subscription_%';
-- log the event
    INSERT INTO SyncLog(event_time, event_name, subs)
    VALUES( CURRENT TIMESTAMP, 'sp_hook_dbmlsync_download_table_end, subs_list );
END
```

Related Information

Synchronization Event Hook Sequence [page 234]

1.6.6.14 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	The exit code for the synchronization just completed. A value other than zero represents a synchronization error.
publication_`n` (in)	publication	Deprecated. Use subscription_`n` instead. The publications being synchronized, where `n` is an integer. There is one publication_`n` entry for each publication being synchronized. The numbering of `n` starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.

Name	Value	Description
upload status (in)	*not sent* | *committed* | *failed* | *unknown*	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. This can occur during a download-only synchronization, a restarted download, or a file-based download. • *committed* - The upload was received by the MobiLink server, and committed. • *failed* - The MobiLink server did not commit the upload. For a transactional upload, the upload status is 'failed' when some but not all the transactions were successfully uploaded and acknowledged by the server. • *unknown* - The upload was not acknowledged by the MobiLink server. There is no way to know if it was committed or not.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
restartable download (in)	*true*|*false*	If *true*, the download for the current synchronization failed and can be restarted. If *false*, the download was successful or it cannot be restarted.
restartable download size (in)	integer	When the restartable download parameter is *true*, this parameter indicates the number of bytes that were received before the download failed. When restartable download is *false*, this value is meaningless.
error hook user state (in)	integer	This value contains information about errors and can be sent from the hooks sp_hook_dbmlsync_all_error, sp_hook_dbmlsync_communication_error, sp_hook_dbmlsync_misc_error, or sp_hook_dbmlsync_sql_error.

Name	Value	Description
*subscription*_n (in)	subscription name(s)	The names of subscriptions being synchronized where n is an integer. This is one subscription_n entry for each subscription being synchronized. The numbering of n starts at zero.

Remarks

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

If an sp_hook_dbmlsync_end hook is defined so that the hook always sets the restart parameter to *sync*, and you specify multiple subscriptions on the dbmlsync command line in the form -s sub1, -s sub2, and so on, then dbmlsync repeatedly synchronizes the first publication and never synchronizes the second.

Actions of this procedure are committed immediately after execution.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

Example

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;
```

Related Information

Synchronization Event Hook Sequence [page 234]
Resumption of Failed Downloads
Error and Warning Handling in Event Hook Procedures [page 239]

1.6.6.15 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	Deprecated. Use subscription_`n` instead. The publications being synchronized, where `n` is an integer. There is one publication_`n` entry for each publication being synchronized. The numbering of `n` starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
discarded storage (in)	number	The number of bytes of discarded memory after the last synchronization.
rescan (in|out)	*true* | *false*	If set to True by the hook, dbmlsync performs a complete rescan before the next synchronization. On entry, this value is set to False.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
*subscription*_`n` (in)	subscription name(s)	The names of subscriptions being synchronized where `n` is an integer. This is one subscription_`n` entry for each subscription being synchronized. The numbering of `n` starts at zero.

Remarks

When more than one -n option or -s option is specified in the command line, dbmlsync may experience fragmentation which results in discarded memory. The discarded memory can be recovered by rescanning the database transaction log. This hook allows you to decide if dbmlsync should rescan the database transaction log to recover memory.

When no other condition has been met that would force a rescan, this hook is called immediately after the sp_hook_dbmlsync_process_exit_code hook.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

Example

The following example cause a log scan if the discarded storage is greater than 1000 bytes.

```
CREATE PROCEDURE sp_hook_dbmlsync_log_rescan ()
BEGIN
 IF EXISTS(SELECT * FROM #hook_dict
  WHERE name = 'Discarded storage' AND value>1000)
 THEN
  UPDATE #hook_dict SET value ='true' WHERE name='Rescan';
 END IF;
END;
```

Related Information

HoverRescanThreshold (hrt) Extended Option [page 176]

1.6.6.16 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 progress value for each subscription being synchronized. The progress value is the offset in the transaction log up to which all data for the subscription has been uploaded. There is one value for each subscription being synchronized. The numbering of n starts at zero. This value matches subscription_n. For example, log offset_0 is the offset for subscription_0.
log scan retry (in)	*true* | *false*	If this is the first time the transaction log has been scanned for this synchronization, the value is false; otherwise it is true. The log is scanned twice when the MobiLink server and dbmlsync have different information about where the scanning should begin.
*publication*_n (in)	publication	Deprecated. Use subscription_n instead. The publications being synchronized, where n is an integer. There is one publication_n entry for each publication being synchronized. The numbering of n starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
*subscription*_n (in)	subscription name(s)	The names of subscriptions being synchronized where n is an integer. This is one subscription_n entry for each subscription being synchronized. The numbering of n starts at zero.

Remarks

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

This hook is ideal for making any last minute changes to the tables being synchronized that you want to include in the upload.

Actions of this procedure are committed immediately after execution.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

Example

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
   "event_id"           integer NOT NULL DEFAULT autoincrement ,
   "event_time"         timestamp NULL,
   "event_name"         varchar(128) NOT NULL ,
   "subs"               varchar(1024) NULL ,
    PRIMARY KEY ("event_id")
)
```

The following logs the beginning of the log scan for each synchronization.

```
CREATE PROCEDURE sp_hook_dbmlsync_logscan_begin ()
BEGIN

    DECLARE subs_list VARCHAR(1024);
-- build a list of subscriptions being synchronized
    SELECT LIST(value) INTO subs_list
     FROM #hook_dict
     WHERE name LIKE 'subscription_%';
-- log the event
    INSERT INTO SyncLog(event_time, event_name, subs)
    VALUES( CURRENT TIMESTAMP, 'sp_hook_dbmlsync_logscan_begin', subs_list );
END
```

Related Information

Synchronization Event Hook Sequence [page 234]

1.6.6.17 sp_hook_dbmlsync_logscan_end

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

Rows in #hook_dict table

Name	Value	Description
ending log offset (in)	number	The log offset value where scanning ended.
*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	Deprecated. Use subscription_n instead. The publications being synchronized, where n is an integer. There is one publication_n entry for each publication being synchronized. The numbering of n starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.

Name	Value	Description
*subscription*_n (in)	subscription name(s)	The names of subscriptions being synchronized where n is an integer. This is one subscription_n entry for each subscription being synchronized. The numbering of n starts at zero.

Remarks

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

Actions of this procedure are committed immediately after execution.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

Example

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
   "event_id"           integer NOT NULL DEFAULT autoincrement ,
   "event_time"         timestamp NULL,
   "event_name"         varchar(128) NOT NULL ,
   "subs"               varchar(1024) NULL ,
    PRIMARY KEY ("event_id")
)
```

The following logs the end of log scanning for each synchronization.

```
CREATE PROCEDURE sp_hook_dbmlsync_logscan_end ()
BEGIN

    DECLARE subs_list VARCHAR(1024);
-- build a list of subscriptions being synchronized
    SELECT LIST(value) INTO subs_list
     FROM #hook_dict
     WHERE name LIKE 'subscription_%';
-- log the event
    INSERT INTO SyncLog(event_time, event_name, subs)
    VALUES( CURRENT TIMESTAMP, 'sp_hook_dbmlsync_logscan_end', subs_list );
```

```
END
```

Related Information

Synchronization Event Hook Sequence [page 234]

1.6.6.18 sp_hook_dbmlsync_misc_error

Use this stored procedure to process dbmlsync errors which are not categorized as database or communication errors. For example, you can implement the sp_hook_dbmlsync_misc_error hook to log errors or perform a specific action when a specific error occurs.

Rows in #hook_dict table

Name	Value	Description
*publication*_`n` (in)	publication	Deprecated. Use subscription_`n` instead. The publications being synchronized, where `n` is an integer. There is one publication_`n` entry for each publication being synchronized. The numbering of `n` starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
error message (in)	error message text	This is the same text that is displayed in the dbmlsync log.
error id (in)	integer	An ID that uniquely identifies the message. Use this row to identify the error message, as the error message text may change.

Name	Value	Description
error hook user state (in|out)	integer	This value can be set by the hook to pass state information to future calls to the sp_hook_dbmlsync_all_error, sp_hook_dbmlsync_communication_error, sp_hook_dbmlsync_misc_error, sp_hook_dbmlsync_sql_error, or sp_hook_dbmlsync_end hooks. The first time one of these hooks is called, the value of the row is 0. If a hook changes the value of the row, the new value is used in the next hook call.
*subscription*_`n` (in)	subscription name(s)	The names of subscriptions being synchronized where `n` is an integer. This is one subscription_`n` entry for each subscription being synchronized. The numbering of `n` starts at zero.

Remarks

If an error occurs during startup before a synchronization has been initiated, the #hook_dict entries for MobiLink user and Script version are set to an empty string, and no publication_`n` or subscription_`n` rows are set in the #hook_dict table.

The error hook user state row provides a useful mechanism for you to pass information about the nature of the error to the sp_hook_dbmlsync_end hook where you might use that information to decide whether to retry the synchronization.

This procedure executes on a separate connection to ensure that operations it performs are not lost if a rollback is performed on the synchronization connection. If dbmlsync cannot establish a separate connection, the procedure is not called.

Since this hook executes on a separate connection you should use care when accessing tables that are being synchronized in your hook procedure because dbmlsync may have locks on these tables. These locks could cause operations in your hook to fail or to wait indefinitely.

Actions of this procedure are committed immediately after execution.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.

- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

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_misc_error to log all types of error messages.

```
CREATE PROCEDURE sp_hook_dbmlsync_misc_error()
BEGIN
 DECLARE msg VARCHAR(10240);
 DECLARE id INTEGER;
 // get the error message text
 SELECT value INTO msg
  FROM #hook_dict
  WHERE name ='error message';
 // get the error id
 SELECT value INTO id
  FROM #hook_dict
  WHERE name = 'error id';
 // log the error information
 INSERT INTO error_log(err_msg,err_id)
  VALUES (msg,id);
END;
```

To see possible error id values, test run dbmlsync. For example, the following dbmlsync command line references an invalid subscription.

```
dbmlsync -c SERVER=custdb;UID=DBA;PWD=sql -s test
```

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

```
1,19912,
 'Subscription ''test'' not found.'
```

To provide custom error handling, check for the error id 19912 in sp_hook_dbmlsync_misc_error.

```
ALTER PROCEDURE sp_hook_dbmlsync_misc_error()
BEGIN
 DECLARE msg VARCHAR(10240);
 DECLARE id INTEGER;
 // get the error message text
 SELECT value INTO msg
  FROM #hook_dict
  WHERE name ='error message';
 // get the error id
 SELECT value INTO id
  FROM #hook_dict
  WHERE name = 'error id';
 // log the error information
 INSERT INTO error_log(err_msg,err_id)
  VALUES (msg,id);
 IF id = 19912 THEN
  // handle invalid subscription
 END IF;
END;
```

Related Information

Error and Warning Handling in Event Hook Procedures [page 239]
sp_hook_dbmlsync_communication_error [page 247]
sp_hook_dbmlsync_all_error [page 243]
sp_hook_dbmlsync_sql_error [page 291]

1.6.6.19 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`	Deprecated. Use *subscription*_n instead. The publications being synchronized, where n is an integer. There is one *publication*_n entry for each publication being synchronized. The numbering of n starts at zero.
MobiLink user (in)	`MobiLink user name`	The MobiLink user for which you are synchronizing.
script version (in)	`script version name`	The MobiLink script version to be used for the synchronization.
connection address (in|out)	`connection address`	When the hook is invoked, this is the address used in the most recent failed attempt to connect. You can set this value to a new connection address that you want to try. If retry is set to true, this value is used for the next attempt to connect.

Name	Value	Description
connection type (in|out)	`network protocol`	When the hook is invoked, this is the network protocol (such as TCPIP) that was used in the most recent failed attempt to connect. You can set this value to a new network protocol that you want to try. If retry is set to true, this value is used for the next attempt to connect. Network protocols are specified using the CommunicationType (ctp) extended option.
user data (in|out)	`user-defined data`	State information to be used if the next connection attempt fails. For example, you might find it useful to store the number of retries that have occurred. The default is an empty string.
allow remote ahead (in|out)	*true* | *false*	This is true only if the dbmlsync -ra option or the RemoteProgressGreater=on synchronization profile option was specified for this synchronization. By changing the value of this row, you can change the value of the option for the current synchronization only.
allow remote behind (in|out)	*true* | *false*	This is true only if the dbmlsync -ra option or the RemoteProgressLess=on synchronization profile option was specified for this synchronization. By changing the value of this row, you can change the value of the option for the current synchronization only.
retry (in|out)	*true* | *false*	Set this value to true to retry a failed connection attempt. The default is *false*.
*subscription_*`n` (in)	`subscription name(s)`	The names of subscriptions being synchronized where `n` is an integer. This is one *subscription_*`n` entry for each subscription being synchronized. The numbering of `n` starts at zero.

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.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

Example

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 IF value = ''then 0 else cast(value as integer)endif
      INTO idx
      FROM #hook_dict
      WHERE name = 'user data';

    IF idx < 5 THEN
        UPDATE #hook_dict
        SET value = idx +1
        WHERE name = 'user data';

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

The next example uses a table containing connection information. When an attempt to connect fails, the hook tries the next server in the list.

```
CREATE TABLE conn_list (
    label   INTEGER PRIMARY KEY,
    addr  VARCHAR( 128 ),
    type  VARCHAR( 64 )
);
INSERT INTO conn_list
 VALUES ( 1, 'host=server1;port=91', 'tcpip' );
INSERT INTO conn_list
 VALUES ( 2, 'host=server2;port=92', 'http' );
INSERT INTO conn_list
 VALUES ( 3, 'host=server3;port=93', 'tcpip' );
COMMIT;
CREATE PROCEDURE sp_hook_dbmlsync_ml_connect_failed ()
BEGIN
 DECLARE idx INTEGER;
 DECLARE cnt INTEGER;
 SELECT if value = ''then 1 else cast(value as integer)endif
  INTO idx
  FROM #hook_dict
  WHERE name = 'user data';

 SELECT COUNT( label ) INTO cnt FROM conn_list;
```

```
  IF idx <= cnt THEN
    UPDATE #hook_dict
        SET value = ( SELECT addr FROM conn_list WHERE label = idx )
        WHERE name = 'connection address';
    UPDATE #hook_dict
     SET value = (SELECT type FROM conn_list WHERE label=idx)
     WHERE name = 'connection type';

    UPDATE #hook_dict
     SET value = idx +1
         WHERE name = 'user data';

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

Related Information

MobiLink Client Network Protocol Options [page 24]
CommunicationType (ctp) Extended Option [page 167]
-r dbmlsync Option [page 147]

1.6.6.20 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	Deprecated. Use subscription_n instead. The publications being synchronized, where n is an integer. There is one publication_n entry for each publication being synchronized. The numbering of n starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
fatal error (in)	*true* | *false*	True when the hook is called because of an error that causes dbmlsync to terminate.

Name	Value	Description
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.
*subscription_*`n` (in)	subscription name(s)	The names of subscriptions being synchronized where `n` is an integer. This is one subscription_`n` entry for each subscription being synchronized. The numbering of `n` starts at zero.

Remarks

A dbmlsync session can run multiple synchronizations when you specify the -n option or -s option more than once in the command line, when you use scheduling, or when you use the restart parameter in sp_hook_dbmlsync_end. In these cases, if one or more of the synchronizations fail, the default exit code does not indicate which failed. Use this hook to define the exit code for the dbmlsync process based on the exit codes from the synchronizations. This hook can also be used to log exit codes.

If an error occurs during startup before a synchronization has been initiated, the #hook_dict entries for MobiLink user and Script version are set to an empty string, and no publication_`n` or subscription_`n` rows are set in the #hook_dict table.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

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;
```

Related Information

Synchronization Event Hook Sequence [page 234]
sp_hook_dbmlsync_abort [page 240]

1.6.6.21 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	Deprecated. Use subscription_`n` instead. The publications being synchronized, where `n` is an integer. There is one publication_`n` entry for each publication being synchronized. The numbering of `n` starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	name of script version	The script version used for the synchronization.
drop hook (out)	*never* | *always* | *on success*	The values can be: *never* - (the default) Do not drop the sp_hook_dbmlsync_schema_upgrade hook from the database. *always* - After attempting to run the hook, drop the sp_hook_dbmlsync_schema_upgrade hook from the database. *on success* - If the hook runs successfully, drop the sp_hook_dbmlsync_schema_upgrade hook from the database. On success is identical to always if the dbmlsync -eh option is used, or the dbmlsync extended option IgnoreHookErrors is set to true, or the IgnoreHookErrors synchronization profile option is set to on.
subscription_`n` (in)	subscription name(s)	The names of subscriptions being synchronized where `n` is an integer. This is one subscription_`n` entry for each subscription being synchronized. The numbering of `n` starts at zero.

Remarks

This hook is primarily provided for backward compatibility purposes. Unless you are using the ScriptVersion extended option you can safely perform schema changes without using this hook by using the START SYNCHRONIZATION SCHEMA CHANGE statement.

When this hook is implemented, dbmlsync locks the tables being synchronized by default.

This stored procedure is intended for making schema changes to deployed remote databases. Using this hook for schema upgrades ensures that all changes on the remote database are synchronized before the schema is upgraded, which ensures that the database continues to synchronize. When this hook is being used you should not set the dbmlsync extended option LockTables to off.

During any synchronization where the upload was applied successfully and acknowledged by MobiLink, this hook is called after the sp_hook_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. It is safe to commit or rollback in this hook.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

Example

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);
 -- Change the script version used to synchronize
 ALTER SYNCHRONIZATION SUBSCRIPTION sub1
  SET SCRIPT VERSION='v2';
 -- If the schema upgrade is successful, drop this hook:
 UPDATE #hook_dict
  SET value = 'on success'
  WHERE name = 'drop hook';
END;
```

Related Information

Schema Changes in Remote MobiLink Clients [page 72]
START SYNCHRONIZATION SCHEMA CHANGE Statement [MobiLink]

1.6.6.22 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`	Deprecated. Use *subscription*_`n` instead. The publications being synchronized, where `n` is an integer. There is one *publication*_`n` entry for each publication being synchronized. The numbering of `n` starts at zero.
MobiLink user (in)	`MobiLink user name`	The MobiLink user for which you are synchronizing.
extended options (out)	`opt=val;...`	Extended options to add for the next synchronization.
*subscription*_`n` (in)	`subscription name(s)`	The names of subscriptions being synchronized where `n` is an integer. This is one *subscription*_`n` entry for each subscription being synchronized. The numbering of `n` starts at zero.

Remarks

If a procedure of this name exists, it is called one or more times before each synchronization.

Extended options specified by this hook apply only to the synchronization identified by the subscription and MobiLink user entries, and they apply only until the next time the hook is called for the same synchronization.

The Schedule extended option may not be specified using this hook.

Actions of this procedure are committed immediately after execution.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

Example

The following example uses sp_hook_dbmlsync_set_extended_options to specify the Increment extended option. The extended option is only applied if sub1 is synchronizing.

```
CREATE PROCEDURE sp_hook_dbmlsync_set_extended_options ()
BEGIN
IF exists(SELECT * FROM #hook_dict
  WHERE name LIKE 'subscription_%' AND value='sub1')
 THEN
   -- specify the Increment extended option
   UPDATE #hook_dict
          SET value = 'Increment=10K'
    WHERE name = 'extended options';
 END IF;
END;
```

Related Information

Synchronization Event Hook Sequence [page 234]
MobiLink SQL Anywhere Client Extended Options [page 160]
How dbmlsync Resolves Conflicting Options [page 165]

1.6.6.23 sp_hook_dbmlsync_set_ml_connect_info

Use this stored procedure to set the network protocol and network protocol option used to connect to the MobiLink server.

Name	Value	Description
*publication*_n (in)	`publication`	Deprecated. Use *subscription*_n instead. The publications being synchronized, where n is an integer. There is one *publication*_n entry for each publication being synchronized. The numbering of n starts at zero.
MobiLink user (in)	`MobiLink user name`	The MobiLink user for which you are synchronizing.
script version (in)	`script version name`	The MobiLink script version to be used for the synchronization.
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.
*subscription*_n (in)	`subscription name(s)`	The names of subscriptions being synchronized where n is an integer. This is one *subscription*_n entry for each subscription being synchronized. The numbering of n starts at zero.

Remarks

You can use this hook to set the network protocol and network protocol options used to connect to the MobiLink server by changing the value in the connection type and/or connection address rows.

The protocol and options can also be set in the sp_hook_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.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

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
```

Related Information

Synchronization Event Hook Sequence [page 234]
MobiLink Client Network Protocol Options [page 24]
CommunicationType (ctp) Extended Option [page 167]
sp_hook_dbmlsync_set_extended_options [page 285]

1.6.6.24 sp_hook_dbmlsync_set_upload_end_progress

This stored procedure can be used to define an ending progress when a scripted upload subscription is synchronized. This procedure is called only when a scripted upload publication is being synchronized.

Rows in #hook_dict table

Name	Value	Description
generating download exclusion list (in)	TRUE | FALSE	TRUE if no upload will be sent during the synchronization (for example, in a download-only synchronization or when a file-based download is applied). In these cases, the upload scripts are still called and the operations generated are used to identify download operations that change rows that need to be uploaded. When such an operation is found, the download is not applied.
*publication*_`n` (in)	publication	Deprecated. Use subscription_`n` instead. The publications being synchronized, where `n` is an integer. There is one publication_`n` entry for each publication being synchronized. The numbering of `n` starts at zero.
*start progress as timestamp*_`n` (in)	progress as timestamp	The starting progress for each subscription being synchronized expressed as a timestamp, where `n` is the same integer used to identify the subscription.
*start progress as bigint*_`n` (in)	progress as bigint	The starting progress for each subscription being synchronized expressed as a bigint, where `n` is the same integer used to identify the subscription.
script version (in)	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.

Name	Value	Description
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.
end progress (in|out)	timestamp	The hook can modify this row to change the "end progress" and "raw end progress" 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 "end progress is bigint" row. The default value for this row is the current timestamp.
*subscription_*`n` (in)	subscription name(s)	The names of subscriptions being synchronized where `n` is an integer. This is one subscription_`n` entry for each subscription being synchronized. The numbering of `n` starts at zero.

Remarks

For a scripted upload, each time an upload procedure is called it is passed a start progress value and an end progress value. The procedure must return all appropriate operations that occurred during the period defined by those two values. The begin progress value is always the same as the end progress value from the last successful synchronization, unless this is a first synchronization, in which case the begin progress is January 1, 1900, 00:00:00.000. By default, the end progress value is the time when dbmlsync began building the upload.

This hook lets you override the default end progress value. You could define a shorter period for the upload or you could implement a progress tracking scheme based on something other than timestamps (for example, generation numbers).

If "end progress is bigint" is set to true, the end progress must be an integer less than or equal to the number of milliseconds from 1900-01-01 00:00:00 to 9999-12-31 23:59:59:9999, which is 255,611,203,259,999.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

Related Information

Custom Progress Values in Scripted Upload [page 318]
Synchronization Event Hook Sequence [page 234]
Scripted Upload [page 308]

1.6.6.25 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	Deprecated. Use subscription_`n` instead. The publications being synchronized, where `n` is an integer. There is one publication_`n` entry for each publication being synchronized. The numbering of `n` starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
error message (in)	error message text	This is the same text that is displayed in the dbmlsync log.
error id (in)	numeric	An ID that uniquely identifies the message.

Name	Value	Description
error hook user state (in|out)	integer	This value can be set by the hook to pass state information to future calls to the sp_hook_dbmlsync_all_error, sp_hook_dbmlsync_communication_error, sp_hook_dbmlsync_misc_error, sp_hook_dbmlsync_sql_error, or sp_hook_dbmlsync_end hooks. The first time one of these hooks is called, the value of the row is 0. If a hook changes the value of the row, the new value is used in the next hook call.
SQLCODE (in)	SQLCODE	The SQLCODE returned by the database when the operation failed.
SQLSTATE (in)	SQLSTATE value	The SQLSTATE returned by the database when the operation failed.
subscription_`n` (in)	subscription name(s)	The names of subscriptions being synchronized where `n` is an integer. This is one subscription_`n` entry for each subscription being synchronized. The numbering of `n` starts at zero.

Remarks

If an error occurs during startup before a synchronization has been initiated, the #hook_dict entries for MobiLink user and Script version are set to an empty string, and no publication_`n` or subscription_`n` rows are set in the #hook_dict table.

You can identify SQL errors using the SQL Anywhere SQLCODE or the ANSI SQL standard SQLSTATE.

The error hook user state row provides a useful mechanism for you to pass information about the nature of the error to the sp_hook_dbmlsync_end hook where you might use that information to decide whether to retry the synchronization.

This procedure executes on a separate connection to ensure that operations it performs are not lost if a rollback is performed on the synchronization connection. If dbmlsync cannot establish a separate connection, the procedure is not called.

Since this hook executes on a separate connection you should use care when accessing tables that are being synchronized in your hook procedure because dbmlsync may have locks on these tables. These locks could cause operations in your hook to fail or to wait indefinitely.

Actions of this procedure are committed immediately after execution.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

Related Information

Error and Warning Handling in Event Hook Procedures [page 239]
SQL Anywhere Error Messages
SQL Anywhere Error Messages
SQL Anywhere Error Messages Sorted by SQLCODE
sp_hook_dbmlsync_all_error [page 243]
sp_hook_dbmlsync_communication_error [page 247]
sp_hook_dbmlsync_misc_error [page 274]

1.6.6.26 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`	Deprecated. Use *subscription*_`n` instead. The publications being synchronized, where `n` is an integer. There is one *publication*_`n` entry for each publication being synchronized. The numbering of `n` starts at zero.
MobiLink user (in)	`MobiLink user name`	The MobiLink user for which you are synchronizing.
Script version (in)	`script version name`	The MobiLink script version to be used for the synchronization.

Name	Value	Description
*subscription*_n (in)	subscription name(s)	The names of subscriptions being synchronized where n is an integer. This is one *subscription*_n entry for each subscription being synchronized. The numbering of n starts at zero.

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.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

Example

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
   "event_id"           integer NOT NULL DEFAULT autoincrement ,
   "event_time"         timestamp NULL,
   "event_name"         varchar(128) NOT NULL ,
   "subs"               varchar(1024) NULL ,
    PRIMARY KEY ("event_id")
)
```

The following logs the beginning of the upload for each synchronization.

```
CREATE PROCEDURE sp_hook_dbmlsync_upload_begin ()
BEGIN

    DECLARE subs_list VARCHAR(1024);
-- build a list of subscriptions being synchronized
    SELECT LIST(value) INTO subs_list
     FROM #hook_dict
     WHERE name LIKE 'subscription_%';
-- log the event
    INSERT INTO SyncLog(event_time, event_name, subs)
    VALUES( CURRENT TIMESTAMP, 'sp_hook_dbmlsync_upload_begin', subs_list );
```

```
END
```

Related Information

Synchronization Event Hook Sequence [page 234]

1.6.6.27 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 the Remarks section.	The cause of failure of an upload. For more information, see Remarks.
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* - The upload was not acknowledged by the MobiLink server. There is no way to know if it was committed or not.

Name	Value	Description
*publication*_n (in)	`publication`	Deprecated. Use *subscription*_n instead. The publications being synchronized, where n is an integer. There is one *publication*_n entry for each publication being synchronized. The numbering of n starts at zero.
MobiLink user (in)	`MobiLink user name`	The MobiLink user for which you are synchronizing.
script version (in)	`script version name`	The MobiLink script version to be used for the synchronization.
authentication value (in)	`value`	This value indicates the results of dbmlsync's attempt to authenticate to the MobiLink server. It 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 re-sent because of a conflict between the log offsets stored in the remote and consolidated databases.
*subscription*_n (in)	`subscription name(s)`	The names of subscriptions being synchronized where n is an integer. This is one *subscription*_n entry for each subscription being synchronized. The numbering of n starts at zero.

Remarks

If a procedure of this name exists, it is called immediately after dbmlsync has sent the upload and received confirmation of it from the MobiLink server.

When performing a transactional upload or an incremental upload this hook is called after each segment of the upload is sent. In these cases, the upload status will be "unknown" each time the hook is called except for the last time.

Actions of this procedure are committed immediately after execution.

The range of possible parameter values for the failure cause row in the #hook_dict table includes:

UPLD_ERR_INVALID_USERID_OR_PASSWORD

The user ID or password was incorrect.

UPLD_ERR_USERID_OR_PASSWORD_EXPIRED

The user ID or password expired.

UPLD_ERR_REMOTE_ID_ALREADY_IN_USE

The remote ID was already in use.

UPLD_ERR_SQLCODE_n

Here, `n` is an integer. A SQL error occurred in the consolidated database. The integer specified is the SQLCODE for the error encountered.

UPLD_ERR_USER_ABORT_REQUEST

The upload was aborted at the user's request.

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

Example

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
   "event_id"           integer NOT NULL DEFAULT autoincrement ,
   "event_time"         timestamp NULL,
   "event_name"         varchar(128) NOT NULL ,
   "subs"               varchar(1024) NULL ,
    PRIMARY KEY ("event_id")
)
```

The following logs the end up the upload for each synchronization.

```
CREATE PROCEDURE sp_hook_dbmlsync_upload_end ()
BEGIN

    DECLARE subs_list VARCHAR(1024);
-- build a list of subscriptions being synchronized
    SELECT LIST(value) INTO subs_list
     FROM #hook_dict
     WHERE name LIKE 'subscription_%';
-- log the event
    INSERT INTO SyncLog(event_time, event_name, subs)
    VALUES( CURRENT TIMESTAMP, 'sp_hook_dbmlsync_upload_end', subs_list );
END
```

Related Information

Synchronization Event Hook Sequence [page 234]

1.6.6.28 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`	Deprecated. Use *subscription*_`n` instead. The publications being synchronized, where `n` is an integer. There is one *publication*_`n` entry for each publication being synchronized. The `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.

Name	Value	Description
*file generation number*_`n` (in)	`number`	The generation numbers from the download file. There is one *file generation number*_`n` for each *subscription*_`n` entry. The `n` in *subscription*_`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 or the DnldFileExtra synchronization profile option.
apply file (in|out)	*True*|*False*	If true (the default), the download file is applied only if it passes dbmlsync's other validation checks. If false, the download file is not applied to the remote database.
check generation number (in|out)	*True*|*False*	If true (the default), dbmlsync validates generation numbers. If the generation numbers in the download file do not match those in the remote database, dbmlsync does not apply the download file. If false, dbmlsync does not check generation numbers.
setting generation number (in)	*true* | *false*	True if the -bg option or UpdateGenNum synchronization profile option was used when the download file was created. When true, the generation numbers on the remote database are updated from the download file and normal generation number checks are not performed.
*subscription*_`n` (in)	`subscription name(s)`	The names of subscriptions being synchronized where `n` is an integer. This is one *subscription*_`n` entry for each subscription being synchronized. The numbering of `n` starts at zero.

Remarks

Use this stored procedure to implement custom checks to decide if a download file can be applied.

To compare the generation numbers or timestamps contained in the file with those stored in the remote database, they can be queried from the SYSSYNC system view.

This hook is called before a file-based download is applied to the remote database.

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

Privileges

Hook procedures can be created by any user with the MANAGE REPLICATION system privilege. However, to ensure that the hook can access the #hook_dict table, which is used to pass information in and out of hooks, hooks must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Be defined using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

Example

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;
```

Related Information

MobiLink File-based Download
-be dbmlsync Option [page 124]
-bg dbmlsync Option [page 125]

1.6.7 Dbmlsync C++ API Reference

The Dbmlsync C++ API provides a programming interface that allows MobiLink client applications written in C++ to launch synchronizations and receive feedback about the progress of the synchronizations they request.

Header file

```
dbmlsynccli.hpp
```

Example

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 practice to check the return value from each API call.

```
#include <stdio.h>
#include "dbmlsynccli.hpp"
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 not required unless
    // your SQL Anywhere install is not in your path or you have multiple
    // versions of the product installed.
    client->SetProperty( "server path", "C:\\SQLAnywhere\\bin32" );
    client->StartServer( 3426,
              "-c server=remote;dbn=rem1;uid=dba;pwd=passwd -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 );
}
```

1.6.8 Dbmlsync .NET API Reference

The Dbmlsync .NET API provides a programming interface that allows MobiLink client applications written in .NET to launch synchronizations and receive feedback about the progress of the synchronizations they request.

Namespace

```
Sap.MobiLink.Client
```

The dbmlsync .NET API reference is available in the *MobiLink - Dbmlsync .NET API Reference* at https://help.sap.com/viewer/63256cabfb224813ac3854b94c7f0bdf/LATEST/en-US.

Example

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.

```
using System;
using System.Collections.Generic;
using System.Text;
using Sap.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 not required unless
            // your SQL Anywhere install is not in your path or you have multiple
            // versions of the product installed.
            cli1.SetProperty("server path", "d:\\sap\\sqlany17\\bin32;"
            cli1.StartServer(3426,
             "-c server=cons;dbn=rem1;uid=dba;pwd=passwd -ve+ -ot c:\\temp\
\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();
        }
    }
}
```

1.6.9 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.

> **i Note**
>
> The Dbmlsync API is the preferred interface for integrating synchronization into your applications. It provides functionality that is very similar to the DBTools interface and is easier to use.

You can use the DBTools interface for dbmlsync to integrate synchronization functionality into your MobiLink synchronization client applications. For example, you can use the interface launch synchronizations and to display dbmlsync output messages in a custom user interface.

The DBTools interface for dbmlsync consists of the following elements that let you configure and run the MobiLink synchronization client:

a_sync_db structure

This structure holds settings, corresponding to dbmlsync command line options, that allow you to customize synchronization. This structure also contains pointers to callback functions that receive synchronization and progress information.

a_syncpub structure

This structure holds publication or subscription information. You can specify a linked list of publications or subscriptions for synchronization.

DBSynchronizeLog function

This function starts the synchronization process. Its only parameter is a pointer to an a_sync_db instance.

In this section:

Related Information

Database Tools Programming
Dbmlsync API [page 113]

1.6.9.1 Setting up the DBTools Interface for dbmlsync

This task guides you through the basic steps to configure and start dbmlsync using the DBTools interface in C or C++.

Procedure

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");
     }
     ```
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 the a_sync_db structure fields.

     ```
     memset( &dbsync_info, 0, sizeof( dbsync_info ) );
     ```
   - Set the 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=passwd";
```

- Set the other a_sync_db fields to customize synchronization.
  Most fields correspond to dbmlsync command line options.
  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;
```

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 example, create a function called dbsyncErrorCallBack to handle generated error messages:

```
extern short _callback dbsyncErrorCallBack( char *str )
{
    if( str != NULL ) {
        printf( "Error Msg    %s\n", str );
    }
    return 0;
}
```

   - For example, create a function called dbsyncWarningCallBack to handle generated warning messages:

```
extern short _callback dbsyncWarningCallBack( char *str )
{
    if( str != NULL ) {
        printf( "Warning Msg  %s\n", str );
    }
    return 0;
}
```

   - For example, create a function called dbsyncLogCallBack to receive verbose informational messages that you might choose to log to a file instead of displaying in a window:

```
extern short _callback dbsyncLogCallBack( char *str )
{
    if( str != NULL ) {
        printf( "Log Msg      %s\n", str );
    }
    return 0;
}
```

   - For example, create a function called dbsyncMsgCallBack to receive informational messages generated during synchronization.

```
extern short _callback dbsyncMsgCallBack( char *str )
{
    if( str != NULL ) {
        printf( "Display Msg  %s\n", str );
    }
    return 0;
}
```

- For example, create a function called dbsyncProgressMessageCallBack to receive the progress text. In the dbmlsync utility, this text is displayed directly above the progress bar.

  ```
  extern short _callback dbsyncProgressMessageCallBack(
   char *str )
  {
      if( str != NULL ) {
          printf( "ProgressText %s\n", str );
      }
      return 0;
  }
  ```
- For example, create a function called dbsyncProgressIndexCallBack to receive information for updating a progress indicator or progress bar. This function receives two parameters:

  index

  An integer representing the current progress of a synchronization.

  max

  The maximum progress value. If this value is zero, the maximum value has not changed since the last time the event was fired.

  ```
  extern short _callback dbsyncProgressIndexCallBack
  (a_sql_uint32 index, a_sql_uint32 max )
  {
      printf( "ProgressIndex    Index %d Max: %d\n",
           index, max );
      return 0;
  }
  ```

  A typical sequence of calls to this callback is shown below:

  ```
  // example calling sequence
  dbsyncProgressIndexCallBack( 0, 100 );
  dbsyncProgressIndexCallBack( 25, 0 );
  dbsyncProgressIndexCallBack( 50, 0 );
  dbsyncProgressIndexCallBack( 75, 0 );
  dbsyncProgressIndexCallBack( 100, 0 );
  ```

  This sequence should result in the progress bar being set to 0% done, 25% done, 50% done, 75% done, and 100% done.
- For example, create a function called dbsyncWindowTitleCallBack to receive status information. In the dbmlsync utility, this information is displayed in the title bar.

  ```
  extern short _callback dbsyncWindowTitleCallBack(
   char *title )
  {
      printf( "Window Title     %s\n", title );
      return 0;
  }
  ```
- The dbsyncMsgQueueCallBack function is called when a delay or sleep is required. It must return one of the following values, which are defined in `dllapi.h`.

  MSGQ_SLEEP_THROUGH

  indicates that the routine slept for the requested number of milliseconds.

  MSGQ_SHUTDOWN_REQUESTED

  indicates that you would like the synchronization to terminate as soon as possible.

MSGQ_SYNC_REQUESTED

indicates that the routine slept for less than the requested number of milliseconds and that the next synchronization should begin immediately if a synchronization is not currently in progress.

```
extern short _callback dbsyncMsgQueueCallBack(
   a_sql_uint32 sleep_period_in_milliseconds )
{
 printf( "Sleep %d ms\n", sleep_period_in_milliseconds );
 Sleep( sleep_period_in_milliseconds );
 return MSGQ_SLEEP_THROUGH;
}
```

- Assign callback function pointers to the appropriate a_sync_db synchronization structure fields.

```
// set call back functions
dbsync_info.errorrtn    = dbsyncErrorCallBack;
dbsync_info.warningrtn  = dbsyncWarningCallBack;
dbsync_info.logrtn      = dbsyncLogCallBack;
dbsync_info.msgrtn      = dbsyncMsgCallBack;
dbsync_info.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 subscriptions should be synchronized.

   Each node in the linked list corresponds to one instance of the -s 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 subscriptions you want to synchronize.
     For example, to synchronize the template_p1 and template_p2 subscriptions together in a single synchronization session:

```
publication_info.next = NULL; // linked list terminates
publication_info.subscription = "template_p1,template_p2";
publication_info.ext_opt  = "dir=c:\\logs";
publication_info.alloced_by_dbsync = 0;
publication_info.pub_name = NULL;
```

     This is equivalent to specifying `-s template_p1,template_p2` on the dbmlsync command line. Specifying extended options using the ext_opt field, provides the same functionality as the dbmlsync -eu option.
   - Assign the publication structure to the upload_defs field of your a_sync_db instance.

```
dbsync_info.upload_defs   = &publication_info;
```

   You can create a linked list of a_syncpub structures. Each a_syncpub instance in the linked list is equivalent to one specification of the -n or -s option on the dbmlsync command line.
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.

```
DBToolsFini( &info );
```

The DBToolsFini function frees DBTools resources.

Results

Dbmlsync is configured and started.

Related Information

Database Tools Programming
DBTools Import Libraries
DBTools Library Initialization and Finalization
DBTools Library Initialization and Finalization
Callback Functions
-c dbmlsync Option [page 127]
-eu dbmlsync Option [page 136]
-s dbmlsync Option [page 148]
-n dbmlsync Option (Deprecated) [page 140]

1.6.10 Scripted Upload

Scripted upload applies only to MobiLink applications that use SQL Anywhere remote databases.

⚠ Caution

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 database is not lost.

However, in some cases you may want to ignore the transaction log and define the upload. Using scripted upload you can define exactly what data you want to upload. When doing scripted upload you do not have to maintain a transaction log for your remote database. Transaction logs take up space that may be at a premium on small devices. However, transaction logs are very important for database backup and recovery, and improve database performance.

To implement scripted upload, you create a special kind of publication that specifies the names of stored procedures that you create. The stored procedures define an upload by returning result sets that contain the rows to insert, update, or delete on the consolidated database.

Note: Do not confuse scripted upload with upload scripts. Upload scripts are MobiLink event scripts on the consolidated database that you write to tell the MobiLink server what to do with the upload. When you use scripted upload, you still need to write upload scripts to apply uploads to the consolidated database and download scripts to determine what to download.

Scenarios

The following are some scenarios where scripted upload may be useful:

- Your remote database is running on a device with limited storage and there is not enough space for a transaction log.
- You want to upload all the data from all your remote databases to create a new consolidated database.
- You want to write custom logic to determine which changes are uploaded to the consolidated database.

Warnings

Before implementing scripted upload, be sure to read this entire section. In particular, take note of the following points:

- If you do not set up your scripted upload correctly, you can lose data.
- When you implement scripted upload, you need to maintain or reference things that dbmlsync normally handles for you. These include pre- and post-images of data, and the progress of the synchronization.
- You need to lock tables on the remote database while synchronizing via scripted upload. With log-based synchronization, locking is not required.
- Transactional uploads are extremely difficult to implement with scripted upload.

In this section:

This tutorial shows you how to set up a scripted upload that provides conflict detection. You create the consolidated and remote databases, stored procedures, publications and subscriptions that are required by scripted upload.

1.6.10.1 Scripted Upload Setup

The following steps provide an overview of the tasks required to set up scripted upload, assuming that you already have MobiLink synchronization set up.

> ⚠ Caution
>
> 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.

Create stored procedures that identify the rows to upload

You can define three stored procedures per table: one each for upload, insert, and delete.

Create a publication that contains the following

The keywords WITH SCRIPTED UPLOAD and the names of the stored procedures.

Related Information

Stored Procedures for Scripted Upload [page 316]
Publications for Scripted Upload [page 322]
Tutorial: Using Scripted Upload [page 322]
LockTables (lt) Extended Option [page 179]

1.6.10.2 Design Considerations for Scripted Upload

Keep the following design considerations in mind 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 the tutorial for using scripted upload.

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.

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.

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 the documentation for performing scripted upload with no table locking.

Redundant uploads

Typically, you want to upload each operation on the remote database exactly once. To help you with this, MobiLink maintains a progress value for each subscription. By default the progress value is the time at which 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.

Each time one of your upload procedures is called, values are passed to it through the #hook_dict table. Among these are the 'start progress' and 'end progress' values. These define the period of time for which the upload being built should include changes to the remote database. Operations that occurred before the 'start progress' have already been uploaded. Those that occur after the 'end progress' should be uploaded during the next synchronization.

Unknown Upload Status

A common mistake in the implementation of scripted upload is creating stored procedures that depend on knowing whether an upload was successfully applied to the consolidated database in the sp_hook_dbmlsync_upload_end or sp_hook_dbmlsync_end hooks. This approach is unreliable.

For example, the following example tries to handle inserts by using a bit on each row to keep track of whether the row needs to be uploaded. The bit is set when a row is inserted, and it is cleared in the sp_hook_dbmlsync_upload_end hook when the upload is successfully committed.

```
//
// DO NOT DO THIS!
//
CREATE TABLE t1 (
   pk          integer primary key,
   val         varchar( 256 ),
   to_upload   bit DEFAULT 1
);
CREATE PROCEDURE t1_ins()
RESULT( pk integer, val varchar(256) )
BEGIN
    SELECT pk, val
    FROM t1
    WHERE to_upload = 1;
END;
CREATE PROCEDURE sp_hook_dbmlsync_upload_end()
BEGIN
    DECLARE     upload_status   varchar(256);
    SELECT value
    INTO upload_status
    FROM #hook_dict
    WHERE name = 'upload status';
    if upload_status = 'committed' THEN
        UPDATE t1 SET to_upload = 0;
    END IF
END;
CREATE PUBLICATION p1 WITH SCRIPTED UPLOAD (
     TABLE t1 USING ( PROCEDURE t1_ins FOR UPLOAD INSERT )
);
```

This approach works most of the time. It fails when a hardware or software failure occurs that stops dbmlsync after the upload has been sent but before it has been acknowledged by the server. In that case, the upload may be applied to the consolidated database but the sp_hook_dbmlsync_upload_end hook is not called and the to_upload bits are not cleared. As a result, in the next synchronization, inserts are uploaded for rows that have already been uploaded. Usually this causes the synchronization to fail because it generates a duplicate primary key error on the consolidated database.

The other case where problems can occur is when communication with the MobiLink server is lost after the upload is sent but before it has been acknowledged. In this case dbmlsync cannot tell if the upload was successfully applied. Dbmlsync calls the sp_hook_dbmlsync_upload_end hook and sets the upload status to unknown. As the hook is written this prevents it from clearing the to_upload bits. If the upload was not applied by the server, this is correct. However, if the upload was applied then the same problem occurs as in the previous paragraph. In both of these cases, the affected remote database is unable to synchronize again until someone manually intervenes to resolve the problem.

Preventing data loss during download

When using scripted uploads, it is possible for data in the remote database that needs to be uploaded to be overwritten by data being downloaded from the consolidated database. This results in the loss of changes made to the remote database. To prevent this data loss your upload procedures must include all changes that were committed in the remote database before the sp_hook_dbmlsync_set_upload_end_progress hook was called in each upload they build.

The following example shows how data can be lost if you violate this rule:

Time	
1:05:00	A row, R, that exists both in the consolidated and remote databases is updated with some new values, R1, in the remote database and the change is committed.
1:06:00	The row R is updated in the consolidated database to some new values R2 and the change is committed.
1:07:00	A synchronization occurs. The upload scripts are written so that the upload only contains operations committed before 1:00:00. This violates the 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.

In this section:

Scripted Upload with No Table Locking [page 313]
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.

Related Information

Tutorial: Using Scripted Upload [page 322]
INSERT Statement
sp_hook_dbmlsync_set_upload_end_progress [page 289]

1.6.10.2.1 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, 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 in-sert_time column for R is set to 1:04:00.

Time	
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.

Time	
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.

Related Information

How to Set the Isolation Level
sa_transactions System Procedure

1.6.10.3 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.

To ensure that your stored procedure can access the #hook_dict table, the procedure must meet one of the following requirements:

- Be owned by a user with the SELECT ANY TABLE and UPDATE ANY TABLE system privileges.
- Have been created using the SQL SECURITY INVOKER clause of the CREATE PROCEDURE statement.

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 database that need to be uploaded are not overwritten by the download.)
subscription_`n`	*subscription name(s)*	The names of subscriptions being synchronized where `n` is an integer. This is one subscription_`n` entry for each subscription being synchronized. The numbering of `n` starts at zero.
publication_`n`	*publication name*	Deprecated. Use subscription_`n` instead. 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.

In this section:

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.

Stored Procedures for Updates [page 321]
The stored procedure for updates must return a result set that includes two sets of values.

Publications for Scripted Upload [page 322]
To create a scripted upload publication, use the keywords WITH SCRIPTED UPLOAD and specify the stored procedures in the USING clause.

Related Information

#hook_dict Table [page 236]

1.6.10.3.1 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.

Related Information

sp_hook_dbmlsync_set_upload_end_progress [page 289]
sa_convert_ml_progress_to_timestamp System Procedure
sa_convert_timestamp_to_ml_progress System Procedure

1.6.10.3.2 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 the tutorial for using scripted upload.

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
   )
)
```

Related Information

Tutorial: Using Scripted Upload [page 322]

1.6.10.3.3 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, the tutorial for using scripted upload.

The following example creates a table called t1 and a publication called p1. The publication specifies WITH SCRIPTED UPLOAD and registers the stored procedure t1_delete as the delete procedure. In the definition of the t1_delete stored procedure, the result set includes all columns listed in the CREATE PUBLICATION statement but in the order in which the columns were declared in the CREATE TABLE statement.

```
CREATE TABLE t1(
   //The column ordering is taken from here
   pk integer primary key,
   c1 char( 30),
   c2,   float,
   c3 double );
CREATE PROCEDURE t1_delete ()
RESULT( pk integer, c1 char(30), c3 double )
begin
   ...
end
CREATE PUBLICATION p1 WITH SCRIPTED UPLOAD(
   // Order of columns here is ignored
   TABLE t1( c3, pk, c1 ) USING (
     PROCEDURE t1_delete FOR UPLOAD DELETE
   )
)
```

Related Information

Tutorial: Using Scripted Upload [page 322]

1.6.10.3.4 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).

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 the tutorial for using scripted upload.

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. 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
   )
)
```

Related Information

Tutorial: Using Scripted Upload [page 322]

1.6.10.3.5 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
      )
   )
```

Related Information

CREATE PUBLICATION Statement [MobiLink] [SQL Remote]
ALTER PUBLICATION Statement [MobiLink] [SQL Remote]

1.6.10.4 Tutorial: Using Scripted Upload

This tutorial shows you how to set up a scripted upload that provides conflict detection. You create the consolidated and remote databases, stored procedures, publications and subscriptions that are required by scripted upload.

Prerequisites

You must have the following roles and privileges:

- SYS_REPLICATION_ADMIN_ROLE system role
- SYS_RUN_REPLICATION_ROLE system role

- CREATE ANY TRIGGER system privilege

1. Lesson 1: Creating the Consolidated Database [page 323]
   In this lesson, you create a consolidated database and set it up to be used with MobiLink.
2. Lesson 2: Creating the Remote Database [page 325]
   In this lesson, you create a remote database and populate it with objects.
3. Lesson 3: Handling Inserts [page 326]
   You must define stored procedures and other things to handle the upload. You do this separately for inserts, deletes, and updates.
4. Lesson 4: Handling Updates [page 328]
   In this lesson, you create a table, a trigger and a stored procedure to handle updates.
5. Lesson 5: Handling Deletes [page 330]
   In this lesson, you create a table, a trigger and a stored procedure to handle deletes.
6. Lesson 6: Clearing Out the Pre-image and Delete Tables [page 332]
   In this lesson, you create an upload_end hook to clear out the pre-image and delete tables.
7. Lesson 7: Creating a Publication, MobiLink User, and Subscription [page 333]
   In this lesson, you create a publication, a MobiLink user and a subscription.
8. Lesson 8: Demonstrating the Scripted Upload [page 334]
   In this lesson, you run SQL statement and commands to demonstrate a scripted upload.
9. Lesson 9: Cleaning Up [page 335]
   Remove the tutorials materials from your computer.

1.6.10.4.1 Lesson 1: Creating the Consolidated Database

In this lesson, you create a consolidated database and set it up to be used with MobiLink.

Prerequisites

You must have the roles and privileges listed at the beginning of this tutorial.

Context

This tutorial specifies file names and assumes they are in the current directory, which is `scriptedupload`. In a real application, you should specify the full path to the file.

This tutorial 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.

Procedure

1. Run the following command to create a directory to hold the tutorial files, and switch to that directory.

   ```
   md c:\scriptedupload
   cd c:\scriptedupload
   ```

2. Run the following command to create a consolidated database:

   ```
   dbinit -dba DBA,passwd consol.db
   ```

3. Next, run the following command to define an ODBC data source for the consolidated database:

   ```
   dbdsn -w dsn_consol -y -c "UID=DBA;PWD=passwd;DBF=consol.db;SERVER=consol"
   ```

4. 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" "%SQLANY17%\MobiLink\setup\syncsa.sql"
   ```

5. To open Interactive SQL and connect to `consol.db` using the dsn_consol DSN, run the following command:

   ```
   dbisql -c "DSN=dsn_consol"
   ```

6. Run the following SQL statements. They create the employee table on the consolidated database, insert values into the table, and create the required synchronization scripts.

   ```
   CREATE TABLE employee (
      id      unsigned integer primary key,
      name    varchar( 256),
      salary  numeric( 9, 2 )
   );
   INSERT INTO employee VALUES( 100, 'smith', 225000 );
   COMMIT;
   CALL ml_add_table_script( 'default', 'employee', 'upload_insert',
          'INSERT INTO employee ( id, name, salary ) VALUES ( {ml r.id}, {ml
   r.name}, {ml r.salary} )' );
   CALL ml_add_table_script( 'default', 'employee', 'upload_update',
          'UPDATE employee SET name = {ml r.name}, salary = {ml r.salary} WHERE
   id = {ml r.id}' );
   CALL ml_add_table_script( 'default', 'employee', 'upload_delete',
          'DELETE FROM employee WHERE id = {ml r.id}' );
   CALL ml_add_table_script( 'default', 'employee', 'download_cursor',
          'SELECT * from employee' );
   ```

   After you have executed the SQL, leave Interactive SQL running and connected to the consolidated database as you will be running more SQL against the database as you work through the tutorial.

Results

The consolidated database is created and set up to be used with MobiLink.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Using Scripted Upload [page 322]

Next task: Lesson 2: Creating the Remote Database [page 325]

1.6.10.4.2 Lesson 2: Creating the Remote Database

In this lesson, you create a remote database and populate it with objects.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

1. At a command prompt in your samples directory, run the following command to create a remote database:
```
dbinit -dba DBA,passwd remote.db
```
2. Next, run the following command to define an ODBC data source:
```
dbdsn -w dsn_remote -y -c "UID=DBA;PWD=passwd;DBF=remote.db;SERVER=remote"
```
3. To open Interactive SQL and connect to `remote.db` using the dsn_remote, run the following command:
```
dbisql -c "DSN=dsn_remote"
```
4. 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'
);
```
   After you have executed the SQL, leave Interactive SQL running and connected to the remote database as you will be running more SQL against the database as you work through the tutorial.

Results

The remote database is created.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Using Scripted Upload [page 322]

Previous task: Lesson 1: Creating the Consolidated Database [page 323]

Next task: Lesson 3: Handling Inserts [page 326]

1.6.10.4.3 Lesson 3: Handling Inserts

You must define stored procedures and other things to handle the upload. You do this separately for inserts, deletes, and updates.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

1. Using the instance of Interactive SQL connected to the remote database, create a trigger to set the insert_time on each row when it is inserted using the following SQL.

```
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;
```

   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.

2. Still in the remote database, create a procedure to return as a result set all the inserted rows to be uploaded.

```
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;
```

Results

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.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Using Scripted Upload [page 322]

Previous task: Lesson 2: Creating the Remote Database [page 325]

Next task: Lesson 4: Handling Updates [page 328]

1.6.10.4.4 Lesson 4: Handling Updates

In this lesson, you create a table, a trigger and a stored procedure to handle updates.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Context

To handle uploads, you must ensure that the correct pre-image is used based on the start progress when the upload was built.

Procedure

1. Using the instance of Interactive SQL connected to the remote database, 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 )
);
```

2. 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.

```
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;
```

   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.

3. Next, create an upload procedure to handle updates.

```
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;
```

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. 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.

Results

A table is created to store pre-images, a trigger is created to store pre-images of each row that is updated, and a stored procedure is created to handle the updates.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Using Scripted Upload [page 322]

Previous task: Lesson 3: Handling Inserts [page 326]

Next task: Lesson 5: Handling Deletes [page 330]

1.6.10.4.5 Lesson 5: Handling Deletes

In this lesson, you create a table, a trigger and a stored procedure to handle deletes.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

1. Using the instance of Interactive SQL connected to the remote database, 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
);
```

2. Next, create a trigger to populate the employee_delete table as rows are deleted from the employee table.

```
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;
```

This trigger is not called during download because later you set the dbmlsync extended option FireTriggers to false. This trigger assumes that a deleted row is never reinserted; therefore it does not deal with the same row being deleted more than once.

3. The next SQL statement creates an upload procedure to handle deletes.

```
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;
```

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.

Results

A table is created to store a list of deletes, a trigger is created to populate the employee_delete table as rows are deleted from the employee table, and an upload procedure is created to handle deletes.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Using Scripted Upload [page 322]

Previous task: Lesson 4: Handling Updates [page 328]

Next task: Lesson 6: Clearing Out the Pre-image and Delete Tables [page 332]

1.6.10.4.6 Lesson 6: Clearing Out the Pre-image and Delete Tables

In this lesson, you create an upload_end hook to clear out the pre-image and delete tables.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Context

This tutorial 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.

Procedure

Using the instance of Interactive SQL connected to the remote database, create an upload_end hook to clean up the employee_preimage and employee_delete tables when an upload is successful.

```
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;
```

Results

The pre-image and delete results are removed from the tables when an upload is successful.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Using Scripted Upload [page 322]

Previous task: Lesson 5: Handling Deletes [page 330]

Next task: Lesson 7: Creating a Publication, MobiLink User, and Subscription [page 333]

1.6.10.4.7 Lesson 7: Creating a Publication, MobiLink User, and Subscription

In this lesson, you create a publication, a MobiLink user and a subscription.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

Using the instance of Interactive SQL connected to the remote database, run the following SQL statements. The publication called pub1 uses the scripted upload syntax (WITH SCRIPTED UPLOAD). It creates an article for the employee table, and registers the three stored procedures you just created for use in the scripted upload. It creates a MobiLink user called u1, and a subscription between v1 and pub1. The extended option FireTriggers is set to off to prevent triggers from being fired on the remote database when the download is applied, which prevents downloaded changes from being uploaded during the next synchronization.

```
CREATE PUBLICATION pub1 WITH SCRIPTED UPLOAD (
TABLE employee( id, name, salary ) USING (
   PROCEDURE employee_insert FOR UPLOAD INSERT,
   PROCEDURE employee_update FOR UPLOAD UPDATE,
   PROCEDURE employee_delete FOR UPLOAD DELETE
      )
);
CREATE SYNCHRONIZATION USER u1;
CREATE SYNCHRONIZATION SUBSCRIPTION TO pub1 FOR u1
TYPE 'tcpip'
ADDRESS 'host=localhost'
OPTION FireTriggers='off'
SCRIPT VERSION 'default';
```

Results

A publication, MobiLink user, and subscription are created.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Using Scripted Upload [page 322]

Previous task: Lesson 6: Clearing Out the Pre-image and Delete Tables [page 332]

Next task: Lesson 8: Demonstrating the Scripted Upload [page 334]

1.6.10.4.8 Lesson 8: Demonstrating the Scripted Upload

In this lesson, you run SQL statement and commands to demonstrate a scripted upload.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

1. Using the instance of Interactive SQL connected to the remote database, insert data to synchronize using scripted upload. Run the following SQL statements on the remote database:

   ```
   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;
```

2. At a command prompt, start the MobiLink server:

   ```
   mlsrv17 -c "DSN=dsn_consol" -o mlserver.mls -v+ -dl -zu+
   ```

3. Start a synchronization using dbmlsync:

   ```
   dbmlsync -c "DSN=dsn_remote" -k -uo -o remote.mlc -v+
   ```

4. Run the following SQL statement using the instance of Interactive SQL connected to the remote database to verify that the inserts were uploaded.

   ```
   SELECT * FROM employee
   ```

Results

You should see the values that were inserted at the beginning of this lesson.

Next Steps

Proceed to the next lesson.

Task overview: Tutorial: Using Scripted Upload [page 322]

Previous task: Lesson 7: Creating a Publication, MobiLink User, and Subscription [page 333]

Next task: Lesson 9: Cleaning Up [page 335]

1.6.10.4.9 Lesson 9: Cleaning Up

Remove the tutorials materials from your computer.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

1. Close all instances of Interactive SQL.
2. Close the SQL Anywhere, MobiLink, and synchronization client windows.
3. Delete all tutorial-related ODBC data sources.
    a. Start the ODBC Administrator.

       Run the following command:

       ```
       odbcad32
       ```
    b. Remove the **`dsn_consol`** and **`dsn_remote`** data sources.
4. Navigate to the directory containing your consolidated and remote databases, `c:\scriptedupload`, and delete all the files.

Results

The tutorial materials are removed from your computer.

Task overview: Tutorial: Using Scripted Upload [page 322]

Previous task: Lesson 8: Demonstrating the Scripted Upload [page 334]

1.7 Advanced Feature: MobiLink Template System (mltemplate)

The MobiLink Template system utility (mltemplate) is a productivity tool for MobiLink developers.

The mltemplate utility is useful for large-scale projects with many tables. It reduces the number of source files required for a synchronization project, making it easier for you to make fixes and add new tables as your project grows. The utility is customizable and can be combined with a build tool to help you iteratively develop your synchronization solution.

The mltemplate utility requires that you have a good understanding of MobiLink and experience creating synchronization scripts and other related objects.

The mltemplate utility commands create a project file in JSON and import a database schema into the file. The schema represents the tables being synchronized. The utility generates SQL files by running Handlebars templates (that you customize and create) against the schema objects in the project file. Which templates get applied to which schema objects is governed by the rules that you manually add to the project file. The output SQL files can be run against a consolidated database or a remote database to create synchronization scripts and other synchronization objects.

In this section:

Generate SQL Files for Your Synchronization Solution (mltemplate) [page 337]
: Use mltemplate to generate SQL files to create synchronization scripts and other objects for your synchronization project.

MobiLink Template Utility (mltemplate) [page 339]
: mltemplate is an advanced utility to help create and maintain your remote databases.

The Project File (mltemplate) [page 344]
: The project file contains the database schema definitions, as well as the definitions for rules and variables.

Rules (mltemplate) [page 348]
: Rules specify which template files apply to which schema objects. You manually define rules in the project file, and then run them by running *mltemplate gen*.

Custom Variables (mltemplate) [page 350]
: Create custom variables in the project file and use them in your template files.

Schema Variables (mltemplate) [page 352]
: Schema variables are generated when you import the database schema into your project. Use these variables in your template files.

Handlebars Templates and Helpers (mltemplate) [page 356]
: mltemplate uses Handlebars templates that typically contain MobiLink synchronization scripts. Templates can also contain synchronization tracking schema, such as definitions for shadow tables, system procedures, and triggers.

Output Files (mltemplate) [page 360]
: Organize the generated content from mltemplate by grouping the output by template and/or by rule.

Tutorial: MobiLink Template System Utility (mltemplate) [page 361]
: Use the mltemplate to create a synchronization project that uses SQL Anywhere consolidated and client databases.

Related Information

MobiLink Template Utility (mltemplate) [page 339]

1.7.1 Generate SQL Files for Your Synchronization Solution (mltemplate)

Use mltemplate to generate SQL files to create synchronization scripts and other objects for your synchronization project.

Prerequisites

You must have an existing consolidated or client database to obtain the database schema from.

Context

Run mltemplate from the directory where the project file is located. mltemplate interprets relative paths as being relative to the directory where mltemplate is run, not from the directory where the project exists.

mltemplate always searches its default installation directories for the files containing the default Handlebars templates, Handlebars helpers, and the global schema variable definitions. Rename the default template files when you copy them and customize them for your own purposes.

mltemplate provides default Handlebars templates for only SAP HANA and SAP SQL Anywhere consolidated databases. You can create templates for the SQL Anywhere client database and/or for all other consolidated databases.

mltemplate can be combined with a build tool to a help you iteratively develop your synchronization solution.

Procedure

1. Set up the directories for your synchronization project, such as a working directory and directories for the Handlebars templates.

   The following table describes an example directory structure.

Directory name	Description
`sync-project`	The working directory for the project file. Always run mltemplate from this directory.
`sync-project \templates\cdb`	Directory to contain your Handlebars templates for the consolidated database. mltemplate provides default Handlebars templates for only SAP HANA and SAP SQL Anywhere consolidated databases. These templates are intended as examples and are not guaranteed for production use. You must create templates for the other consolidated databases.
`sync-project \templates\rem`	Directory to contain your Handlebars templates for the client database. You must create these templates.
`sync-project \helpers`	Directory to contain the additional Handlebars helpers that you create. The default helpers are located in *%SQLANY17%*`\MobiLink\MLTemplate\helpers \sqlhelpers.js`.
`sync-project\out`	Directory to contain the output generated from *mltemplate gen*.

2. Create the project file by running *mltemplate new*.

   For example:

   ```
   mltemplate new project.json  -ConsType sqla -t sync-project\templates\cdb
   ```

   The project file is created.
3. Run mltemplate import to import the database schema from the existing consolidated or client database into the project file.

Schema definitions for the tables are added to the project file in the SyncTables array. mltemplate generates schema variables based on the schema of your database. These variables are added at the end of the SyncTables array.

4. In the project file:
    a. Review the schema variables that mltemplate generated. These variables are used in the default templates.
    b. Manually create rules to link the tables to the Handlebars templates.

       Create rules that group tables with similar behavior together and that assign Handlebars templates to those groups.
    c. Manually define custom variables that you can use in the templates.
5. Create Handlebars templates for your synchronization project.
    a. Copy the default Handlebars templates from *%SQLANY17%*`\MobiLink\MLTemplate\Templates\ConsDB\` to your templates directory, `sync-project\templates\cdb`. Add the templates directories to the templateRoots array in the project file.
    b. Rename the copied template files, so that mltemplate doesn't confuse them with the versions in the installation directory.
    c. Customize the template files and create new template files as required.
    d. Reference these template files in the rules in the project file.
6. Create Handlebars helpers, if required, and add the files to the templateHelpers array in the project file.
    a. Create a JavaScript file. Do not name the file the same name as the default helper file, `sqlhelpers.js`.
    b. Create the helpers and register them by including a Handlebars.registerHelper call in the file.
    c. Add the helpers file to the templateHelpers array in the project file.
7. Run *mltemplate gen* to generate SQL files from the project file and the templates.
8. Use the SQL files to create definitions in the consolidated database and/or the remote database.

Related Information

MLTEMPLATE_JARS Environment Variable
MobiLink Template Utility (mltemplate) [page 339]

1.7.2 MobiLink Template Utility (mltemplate)

mltemplate is an advanced utility to help create and maintain your remote databases.

Syntax

```
mltemplate
  new project-filename.json new-options
| import project-filename.json import-options
| gen project-filename.json gen-options
| modify project-filename.json modify-options
```

new

Creates a project file that stores the settings and schema for your synchronization project.

new-options	Description
-consType `database-type`	Specifies the database type of the consolidated database. This option sets the value of the consType field in the project file. This option is required. Specify *SQLA* for SAP SQL Anywhere, *HANA* for SAP HANA, *ASE* for SAP Adaptive Server Enterprise, *DB2* for IBM DB2 (deprecated), *MSS* for Microsoft SQL Server, *MYS* for MySQL, *ORA* for Oracle, and *IQ* for SAP IQ.
-js `javascript-helper-file`	Specifies the JavaScript file that contains your customized template helpers. This option adds the specified template helper files to those listed in the templateHelpers array in the project file. This option can be specified multiple times. By default, the templateHelpers array contains the default template helper file, `sqlhelpers.js`.
-o `output-file`	Specifies the global output file name. This option sets the value of the outFile field in the project file. By default, the generated content is saved to this SQL file. The default value is `generatedScripts.txt`.
-remType `database-type`	Specifies the database type of the remote databases. This option sets the value of remoteDBType field in the project file. Specify *UL* for UltraLite or *SQLA* for SAP SQL Anywhere. The default is *SQLA*.
-s `script-version-string`	Specifies the MobiLink script version of the project. This option sets the value of the scriptVersion field in the project file. Specify a meaningful string for the version.
-t `template-root-directory`	Specifies the directory where your customized Handlebars templates are located. This option adds your customized templates directory to the list in the templateRoots array in the project file. This option is required. This option can be specified multiple times. There are default templates for SAP SQL Anywhere and SAP HANA consolidated databases in the installation directory (*%SQLANY17%*`\MobiLink\MLTemplate\Templates`). You can create a synchronization system with only these templates. But, often you must replace these templates or supplement them by creating your own templates. For the other supported consolidated databases, you must provide the templates.
-y	Silently overwrites existing project files to create this project file.

import

Imports the schema of the consolidated database or an existing remote database. The schema represents the tables being synchronized, and is used by *mltemplate gen*. The database schema is added to the project file in the SyncTables array. Schema variables are generated from the database schema and are to the project file. Column data types are automatically converted between the consolidated and remote databases as part of the import.

import-options	Description
-catalog `catalog-name`	Specifies the catalog to which the tables belong. When this option is not specified, all catalogs are imported.
-cons	Specifies that the database, whose schema you are importing, is the consolidated database. This option is required when you are importing from the consolidated database. Do not specify the -consType option with this option.
-consType `database-type`	Specifies the type of the consolidated database. This information is required for data type conversions. This option updates the consType field in the project file. This option is required when the -rem option is specified and cannot be specified when the -cons option is specified. Specify *SQLA* for SAP SQL Anywhere, *HANA* for SAP HANA, *ASE* for SAP Adaptive Server Enterprise, *DB2* for IBM DB2 (deprecated), *MSS* for Microsoft SQL Server, *MYS* for MySQL, *ORA* for Oracle, and *IQ* for SAP IQ.
-f `file`	Imports the schema for the tables listed in the text file. The tables must be listed one per line in the file.
-jdbc `jdbc-connection-string`	Specifies the JDBC connection string to the database whose schema you are importing. For example: `-jdbc jdbc:sqlanywhere:DBF=Remotedatabase.db;UID=user;PWD=pwd;SERVER=remote`. Specify either -odbc or -jdbc. For SQL Anywhere databases, mltemplate uses the SQL Anywhere JDBC driver. For all other databases, you must provide the path to the JAR file for their JDBC driver. Use the MLTEMPLATE_JARS environment variable to specify the JAR file.
-odbc `odbc-connection-string`	Specifies the ODBC connection string to the database, whose schema you are importing. For example: `-odbc DSN=mydsn;UID=user;PWD=pwd`. Specify either -odbc or -jdbc.
-p `regular-expression`	Imports the schema for the tables whose names match the specified regular expressions.
-schema `schema-name`	Specifies the schema that the table(s) belong to. When this option is not specified, all schemas are imported.
-re	When importing into an existing project file, re-import the schema for tables that are already defined in the project.
-rem	Specifies that the database, whose schema you are importing, is an existing remote database. You must also specify -consType. This option is the default.
-rule { `rule-name` | *none* }	Adds the table schema definitions to the rule specified. The default is to add the definitions to the rule named *default*. When *none* is specified, the definitions are not added to a rule.

import-options	Description
-t `tableName`	Imports the schema for the specified tables. By default, all tables are imported. This option can be specified multiple times.
-xc `columnName`	Exclude column with the given name. This option can be specified multiple times.

gen

Generates SQL files based on the rules that you define in the project file. The rules run Handlebars templates against the tables in the imported database schema. The output is typically SQL files that contain synchronization scripts and other objects that you use to build the databases for your synchronization project.

gen-options	Description
-d `outDir`	Specifies the directory to save the generated SQL files to. This option overrides the outDir field setting in the project file
-dump	Outputs all variables that mltemplate sends to the Handlebars template files. This helps you debug your templates and variables.
-o `out-file`	Specifies the SQL file to save the generated output content to. This option overrides the value specified in outFile field in the project file.
-rule `rule-name`	Generates output for only the specified rule. By default, output is generated for all rules. This option can be specified multiple times.
-table `table-name`	Generates output for only the specified table. By default, output is generated for all tables specified in the project file. This option can be specified multiple times.
-template `template-name`	Generates output for only the specified Handlebars template file. By default, output is generated for all template files specified in the project file. This option can be specified multiple times.
-v `variable-name variable-value`	Defines a custom variable. If there is a variable with the same name in the project, then this variable overrides its definition. The -v option provides an alternative method to defining variables in the project file. For example, you could use the -v option to specify a user name, and then run mltemplate multiple times for different users. This option can be specified multiple times.

modify

Updates the project file based on a specified JavaScript expression or file. The modify command allows you to edit the database schema in the project file. For example, you could run mltemplate modify to change the case of the column names in the project file. The modify command loads the project file into a JavaScript global variable named syncModel, executes the JavaScript expression provided, and then it saves the modified project file.

modify-options	Description
`javascript-expression`	Modifies the project file based on the specified JavaScript expression or file.

Remarks

Run mltemplate from the directory where the project file is located. mltemplate interprets relative paths as being relative to the directory where mltemplate is run, not from the directory where the project exists.

The mltemplate utility always searches its default installation directories for the files containing the default Handlebars templates, Handlebars helpers, and the global schema variable definitions. Rename the default template files when you copy them and customize them for your own purposes.

The mltemplate utility provides default Handlebars templates for only SAP HANA and SAP SQL Anywhere consolidated databases. You must create templates for the SQL Anywhere remote database and for all other consolidated databases.

Example

The following example creates a project file named `project.json` for a synchronization system whose consolidated and remote databases are SQL Anywhere databases.

```
mltemplate new project.json -t template/mobilink/cdb/sqla -s V1 -remType sqla -
consType sqla -y
```

The following example imports the Products table from the SQL Anywhere sample database into the project.

```
mltemplate import project.json -jdbc "jdbc:sqlanywhere:DSN=SQL Anywhere 17
Demo;uid=DBA;pwd=sql" -rem -t Products
```

Related Information

http://handlebarsjs.com/

http://handlebarsjs.com/

The Project File (mltemplate) [page 344]

http://handlebarsjs.com/

Rules (mltemplate) [page 348]

Handlebars Templates and Helpers (mltemplate) [page 356]

Rules (mltemplate) [page 348]

1.7.3 The Project File (mltemplate)

The project file contains the database schema definitions, as well as the definitions for rules and variables.

The project file is a JSON file that is created when you run *mltemplate new*. The project file contains:

- Environmental information about the project, such as the database types of the client and consolidated databases, and the Handlebars template files. This information is located in the top-level fields in the JSON object. To configure this information run *mltemplate new* and *mltemplate import*.
- Table schema definitions that are imported when you run *mltemplate import*. These definitions are located in the SyncTables array along with any related variables or rules that you create. Run *mltemplate import* to configure the database schema definitions. Manually edit the project file to add and manage variables and rules related to the definitions.
- Definitions of the rules and custom variables. To configure the global variable definitions and global rules, manually edit the project file.

Example

An example of the global options in a project file:

```
{
  "outDir": null,
  "outFile": "generatedScripts.txt",
  "remoteDBType": "SQL Anywhere",
  "consDBType": "SQL Anywhere",
  "templateRoots": [
      "template"
  ],
  "templateVariables": {
      "scriptVersion": "V1",
      "consDBTableOwner": "GROUPO",
      "projectName": "sqla_demo"
  },
  "templateHelpers": [
      "sqlhelpers.js",
      "helpers/custom_helpers.js"
  ],
  "templateOutFiles": {},
  "templateRules": [
    {
      "ruleName": "default",
      "outFile": null,
      "tables": [
        "Products"
      ],
      "templateVariables": {},
      "templateFiles": [
          "mobilink/cdb/sqla/basic/shadow_table.hbs",
          "mobilink/cdb/sqla/basic/trigger.hbs",
          "mobilink/cdb/sqla/basic/sync_script.hbs",
          "mobilink/cdb/sqla/basic/stored_procedure.hbs",
          "mobilink/cdb/sqla/basic/load.hbs"
      ]
    },
    { "ruleName": "cdb_report",
      "tables": null,
      "templateVariables": { },
      "templateFiles": [
          "mobilink/cdb/sqla/test/report.hbs"
      ]
    },
    {
        "ruleName": "rdb_tables",
        "outFile": null,
        "tables": [
            "Products"
        ],
        "templateVariables": {},
        "templateFiles": [
            "mobilink/rdb/sqla/table.hbs"
        ]
      },
      { "ruleName": "publication",
        "tables": null,
        "templateVariables": { },
        "templateFiles": [
            "mobilink/rdb/sqla/publication.hbs"
        ]
      },
      { "ruleName": "subscription",
        "tables": null,
        "templateVariables": {
            "ml_host": "localhost",
            "ml_port": "2439"
        },
        "templateFiles": [
            "mobilink/rdb/sqla/subscription.hbs"
        ]
      },
      { "ruleName": "rdb_report",
        "tables": null,
        "templateVariables": { },
        "templateFiles": [
            "mobilink/rdb/sqla/test/report.hbs"
        ]
      },
      { "ruleName": "rdb_insert",
        "tables": null,
        "templateVariables": { },
        "templateFiles": [
            "mobilink/rdb/sqla/test/insert.hbs"
        ]
      }
  ],
```

An example of the syncTables object in a project file.

```
  "syncTables": [
     {
       "tableName": "Products",
       "consDBTableName": "Products",
       "templateVariables": {},
       "columns": [
         {
           "columnName": "ID",
           "templateVariables": {},
           "baseDomain": "INTEGER",
           "domain": "INTEGER",
           "size": null,
           "scale": null,
           "nullable": 0,
           "ordinal": 1,
           "cons": {
             "columnName": "ID",
             "baseDomain": "INTEGER",
             "domain": "INTEGER",
             "size": null,
             "scale": null,
             "nullable": 0
           }
         },
         {
           "columnName": "Name",
           "templateVariables": {},
           "baseDomain": "CHAR",
           "domain": "CHAR(15)",
           "size": 15,
           "scale": null,
           "nullable": 0,
           "ordinal": 2,
           "cons": {
             "columnName": "Name",
             "baseDomain": "CHAR",
             "domain": "CHAR(15)",
             "size": 15,
             "scale": null,
             "nullable": 0
           }
         },
         {
           "columnName": "Description",
           "templateVariables": {},
           "baseDomain": "CHAR",
           "domain": "CHAR(30)",
           "size": 30,
           "scale": null,
           "nullable": 0,
           "ordinal": 3,
           "cons": {
             "columnName": "Description",
             "baseDomain": "CHAR",
             "domain": "CHAR(30)",
             "size": 30,
             "scale": null,
             "nullable": 0
           }
         },
         {
           "columnName": "Size",
           "templateVariables": {},
           "baseDomain": "CHAR",
           "domain": "CHAR(18)",
           "size": 18,
           "scale": null,
           "nullable": 0,
           "ordinal": 4,
           "cons": {
             "columnName": "Size",
             "baseDomain": "CHAR",
             "domain": "CHAR(18)",
             "size": 18,
             "scale": null,
             "nullable": 0
           }
         },
         {
           "columnName": "Color",
           "templateVariables": {},
           "baseDomain": "CHAR",
           "domain": "CHAR(18)",
           "size": 18,
           "scale": null,
           "nullable": 0,
           "ordinal": 5,
           "cons": {
             "columnName": "Color",
             "baseDomain": "CHAR",
             "domain": "CHAR(18)",
             "size": 18,
             "scale": null,
             "nullable": 0
           }
         },
         {
           "columnName": "Quantity",
           "templateVariables": {},
           "baseDomain": "INTEGER",
           "domain": "INTEGER",
           "size": null,
           "scale": null,
           "nullable": 0,
           "ordinal": 6,
           "cons": {
             "columnName": "Quantity",
             "baseDomain": "INTEGER",
             "domain": "INTEGER",
             "size": null,
             "scale": null,
             "nullable": 0
           }
         },
         {
           "columnName": "UnitPrice",
           "templateVariables": {},
           "baseDomain": "NUMERIC",
           "domain": "NUMERIC(15, 2)",
           "size": 15,
           "scale": 2,
           "nullable": 0,
           "ordinal": 7,
           "cons": {
             "columnName": "UnitPrice",
             "baseDomain": "NUMERIC",
             "domain": "NUMERIC(15, 2)",
             "size": 15,
             "scale": 2,
             "nullable": 0
           }
         },
         {
           "columnName": "Photo",
           "templateVariables": {},
           "baseDomain": "LONG BINARY",
           "domain": "LONG BINARY",
           "size": null,
           "scale": null,
           "nullable": 1,
           "ordinal": 8,
           "cons": {
             "columnName": "Photo",
             "baseDomain": "LONG BINARY",
             "domain": "LONG BINARY",
             "size": null,
             "scale": null,
             "nullable": 1
           }
         }
       ],
       "consDBSchema": "GROUPO",
       "consDBCatalog": null,
       "primaryKeyColumns": [
         "ID"
       ],
       "foreignKeys": []
     }
   ]
```

Related Information

MobiLink Template Utility (mltemplate) [page 339]

1.7.4 Rules (mltemplate)

Rules specify which template files apply to which schema objects. You manually define rules in the project file, and then run them by running *mltemplate gen*.

Syntax

```
"templateRules":
[
      {
         "ruleName":  "rule-name",
          "outFile":  "output-filename.sql",
         "tables":  [
                        table-name[,  ...]
                    ],
         "templateVariables":  {
                     variable-name:variable-value[,  ...]
                     }
          "templateFiles":[
                     handlebars-filename.hbs"[,  ...]
                     ]
      }[,...]
]
```

Parameters

rule-name The name of the rule.

output-filename.sql The SQL file to save the output that is generated by the rule. When *null* is specified, the output for the rule is saved to the default output file, which is defined in the top-level outFile field in the project file. Specifying an output file for each rule makes your project easier to manage, especially when you only need to change one rule.

table-name The tables in the schema that the rule applies to. Specify `null` to create a global rule(a rule with no tables associated with it).

variable-name:variable-value

Definitions of the custom variables that are used in the Handlebars template files. These definitions are passed to template files when the rule runs.

handlebars-filename.hbs The list of Handlebars template files that the rule applies.

Remarks

Rules connect the schema objects stored in the project file with the content defined in the template files.

Table rules are defined inside table objects using the templateRules field.

Global rules are rules that do not apply to any tables. Global rules are useful for creating connection scripts, such as the authentication scripts. Set the *tables* field to null to create a global rule.

Example

Example of a rule that applies only to the Products table:

```
 "templateRules": [
    {
      "ruleName": "default",
      "outFile": null,
      "tables": [
        "Products"
      ],
      "templateVariables": {},
      "templateFiles": [
          "mobilink/cdb/sqla/basic/shadow_table.hbs",
          "mobilink/cdb/sqla/basic/trigger.hbs",
          "mobilink/cdb/sqla/basic/sync_script.hbs",
          "mobilink/cdb/sqla/basic/stored_procedure.hbs",
          "mobilink/cdb/sqla/basic/load.hbs"
      ]
    },
...
```

Example of a global rule.

```
 { "ruleName": "cdb_report",
      "tables": null,
      "templateVariables": { },
      "templateFiles": [
          "mobilink/cdb/sqla/test/report.hbs"
      ]
    },
...
```

Example of a global rule that also defines variables for use within the rule.

```
 { "ruleName": "subscription",
        "tables": null,
        "templateVariables": {
            "ml_host": "localhost",
            "ml_port": "2439"
        },
        "templateFiles": [
            "mobilink/rdb/sqla/subscription.hbs"
        ]
      },
...
```

An example of the default rule in a project file

```
{
  "outDir": null,
  "outFile": "generatedScripts.txt",
  "remoteDBType": "SQL Anywhere",
  "consDBType": "SQL Anywhere",
  "templateRoots": [
      "template"
  ],
  "templateVariables": {
      "scriptVersion": "V1",
      "consDBTableOwner": "GROUPO",
      "projectName": "sqla_demo"
  },
  "templateHelpers": [
      "sqlhelpers.js",
      "helpers/custom_helpers.js"
  ],
  "templateOutFiles": {},
  "templateRules": [
    {
      "ruleName": "default",
      "outFile": null,
      "tables": [
        "Products"
      ],
...
```

Related Information

MobiLink Template Utility (mltemplate) [page 339]

1.7.5 Custom Variables (mltemplate)

Create custom variables in the project file and use them in your template files.

Syntax

```
"templateVariables": {
    "variable-name": "variable-value" [ , ... }
```

Parameters

variable-name The name of the variable.

variable-value The value for the variable.

Remarks

Most custom variables are defined in the project file. However, you can define custom variables at generation time using *mltemplate gen* with the -v option. Variables defined with the -v option override the variables defined in the project file.

In the project file, the location of the custom variable specifies its type and scope. The following table lists the variable types for a project file in the order of their precedence from highest to lowest.

Custom variable type	Variable location
Global variables	Variables defined in the templateVariables field at the beginning of the project file. This custom variable type has the highest precedence of all the variables defined in the project file. Override it by running *mltemplate gen* with the -v option.
Per rule variables	Variables defined inside a rule object by using the template-Variables field.
Per table variables	Variables defined inside a table object by the field template-Variables field.
Column object variables	Variables defined inside column objects by using the templa-teVariables field.

When your run *mltemplate gen*, the variables are extracted from the project file and passed as input to the Handlebars templates so that they are available for substitution in your templates.

Run *mltemplate gen* with the -dump option to obtain all the variables that are passed to the Handlebars templates.

Example

The following example runs *mltemplate gen* with the -dump option to produce JSON files that contain the variables that are passed to the Handlebars templates:

```
mltemplate gen project.json -dump
INFO: Applying template snapshot_download_scripts.hbs for table "Products" and
rule "default"
INFO: Template input variables dumped to file:
varDump_snapshot_download_scripts_Products.json
INFO: Applying template upload_scripts.hbs for table "Products" and rule
"default"
INFO: Template input variables dumped to file:
varDump_upload_scripts_Products.json
INFO: Writing output file generatedScripts.txt
```

An example of global variable definitions:

```
{
  "outDir": null,
  "outFile": "generatedScripts.txt",
  "remoteDBType": "SQL Anywhere",
  "consDBType": "SQL Anywhere",
  "templateRoots": [
      "template"
  ],
  "templateVariables": {
   "scriptVersion": "V1",
   "consDBTableOwner": "GROUPO",
   "projectName": "sqla_demo"
 },
  "templateHelpers": [
      "sqlhelpers.js",
      "helpers/custom_helpers.js"
  ],
  ...
```

An example of variables defined within a rule:

```
 { "ruleName": "subscription",
        "tables": null,
        "templateVariables": {
    "ml_host": "localhost",
    "ml_port": "2439"
  },
        "templateFiles": [
            "mobilink/rdb/sqla/subscription.hbs"
        ]
      },
```

Related Information

MobiLink Template Utility (mltemplate) [page 339]
Schema Variables (mltemplate) [page 352]

1.7.6 Schema Variables (mltemplate)

Schema variables are generated when you import the database schema into your project. Use these variables in your template files.

Run *mltemplate import* to generate the schema variables and add them to the syncTables array in the project file. Schema variables are specific to the schema of the database that you import. If a schema variable does not apply to the imported database schema, then the value for that variable is defined as empty.

Run *mltemplate gen* with the -dump option to obtain all the variables that are passed to the Handlebars templates. Each time a template is run, *mltemplate gen* creates a JSON file containing the schema and custom variables that were passed to the templates.

The following tables describe the default schema variables that are available:

Table schema variables	Description
consDBTableName	The name of consolidated database table.
columns	An array of column objects.
primaryKeyColumns	An array of column names. This is expanded to be an array of column objects before passing to a Handlebars template.
foreignKeys	An array of foreign key objects that includes the following fields: • *foreignKeyName* • *primaryTableSchema* • *primaryTableName* • *columns* The columns field contains an array that contains *primaryColumnName* and *foreignColumnName*.
columnNames	A comma-separated list of column names.
consDBColumnNames	A comma-separated list of consolidated column names.
primaryKeyColumnNames	A comma-separated list of all of the primary key column names.
consDBPrimaryKeyColumnNames	A comma-separated list of all of the primary key column names in the consolidated database.
nonPrimaryKeyColumns	An array containing the names of all non-primary key col-umns.
nonPrimaryKeyColumnNames	A comma-separated list of all of the non-primary key column names.
consDBNonPrimaryKeyColumnNames	A comma-separated list of all non-primary key column names from the consolidated database.
dotIfDefined `name`	The name to of the table's owner, schema or catalog (de-pending upon the database type imported). Edit the dotIfDefined variable to specify a different name. The specified name (along with a dot) will appear before the table name in the output.

Table schema variables	Description
nullClause { *null* | *not null* }	This variable allows you to refer to the value of the nullClause field in the Handlebars templates. It is also useful when writing column declarations in a CREATE TABLE statement. All column definitions in the project file include a nullClause field. This field contains the string *null* when the column allows null values; otherwise contains *not null*.

Rule schema variables	Description
ruleTables	An array of table names that are part of the current rule.
ruleName	The name of the rule.
syncTables	The complete content of the syncTables array in the project file.

Extending the schema variables definitions

The definitions of the schema variables are stored in JavaScript functions in *%SQLANY17%*`\MobiLink\JavaScript\prepare.js`. Extend these variables by creating a JavaScript file with the extension functions, and then adding the file to the templateHelpers array in the project file. Do not name your JavaScript file `prepare.js` because mltemplate always searches for this file in the default directory.

Extend the schema variables with one of the following extension functions:

Extension function	Notes
`customGlobalVars( project )`	This function replaces the customGlobalVars function. Include a call to the defaultGlobalVars function in your JavaScript file so that the table schema variables remain available.
`customTableVars( project, table )`	This function overrides the defaultTableVars function. Include a call to the defaultTableVars function in your JavaScript file so that the table schema variables remain available.
`customRuleVars( project, rule )`	This function overrides the defaultRuleVars function. Include a call to the defaultRuleVars function in your JavaScript file so that the table schema variables remain available.

Example

The following JavaScript code overrides the customGlobalVars function to add a new variable named myCustomVar

```
function customGlobalVars( model )
{
    var globalVars = defaultGlobalVars( model );
    globalVars.myCustomVar = "ABCD";
    return globalVars;
}
```

The following example contains four schema variables:

```
...
  "syncTables":
  [
    {
      "tableName": "Products",
      "consDBTableName": "Products",
      "templateVariables": {},
      "columns":
      [
        {
          "columnName": "ID",
          "templateVariables": {},
          "baseDomain": "INTEGER",
          "domain": "INTEGER",
          "size": null,
          "scale": null,
          "nullable": 0,
          "ordinal": 1,
          "cons":
            {
            "columnName": "ID",
            "baseDomain": "INTEGER",
            "domain": "INTEGER",
            "size": null,
            "scale": null,
            "nullable": 0
            }
        },
        ...
       ],
      "consDBSchema": "GROUPO",
   "consDBCatalog": null,
   "primaryKeyColumns": [  "ID"  ],
   "foreignKeys": []
  }
  ]
}
```

Related Information

Custom Variables (mltemplate) [page 350]
MobiLink Template Utility (mltemplate) [page 339]

1.7.7 Handlebars Templates and Helpers (mltemplate)

mltemplate uses Handlebars templates that typically contain MobiLink synchronization scripts. Templates can also contain synchronization tracking schema, such as definitions for shadow tables, system procedures, and triggers.

mltemplate provides default Handlebars templates to create some synchronization scripts for SAP HANA or SAP SQL Anywhere consolidated databases. The default templates are located in the *%SQLANY17%*`\MobiLink\MLTemplate\Templates` directory. Open a template file in a text editor to view its description.

Use these Handlebars templates as guides when creating templates for your synchronization project. You must create customized templates:

- To supplement the default templates for SAP HANA or SAP SQL Anywhere consolidated databases. Often synchronization projects require functionality that isn't available in the default templates. For example, you must create your own template to filter the data that is downloaded to a client database.
- For the SQL Anywhere remote database and for the other supported consolidated database types.

Use the default templates as guides for creating your own templates.

1. Copy the default files from *%SQLANY17%*`\MobiLink\MLTemplate\Templates` to a location where you have read and write access, edit the files, and then rename them. Rename the files or mltemplate could continue to use the default version. mltemplate always searches its default template directory, whether or not the default directory is referenced in a templateRoots array in the project file.
2. Add your templates directory to the top-level templateRoots array in the project file.
3. Add your template files to the templateFiles arrays for the rules in the project file.

Templates can:

- Contain Handlebars scripts for creating database objects such as synchronization scripts.
- Use schema variables that are added to the project file when the database schema is imported.
- Use custom variables that you define in templateVariable arrays in the project file.
- Use built-in helpers provided by Handlebars.
- Use default helpers defined in `sqlhelpers.js` for formatting SQL statements.
- Use custom helpers that you create in a JavaScript file and add to the templateHelpers array in the project file.

Default Handlebars Template Helpers (`sqlhelpers.js`)

The default helpers are defined in `sqlhelpers.js`, which is located in *%SQLANY17%*`\MobiLink\MLTemplate\Helpers`. The templateHelpers array in the project file contains a reference to `sqlhelpers.js`. This reference does not include a path because it is not needed as mltemplate always searches the default `Helpers` directory for the file.

The following table lists the default helpers that are defined in `sqlhelpers.js`:

Handlebar helper	Description
commaList	Takes an array and separates the block text of each element with commas. This helper is useful when composing a list of column parameters.
orList	Takes an array and separates the block text of each element with logical OR operators. This helper is useful when composing a WHERE clause predicate that is based on a list of columns.
andList	Takes an array and separates the block text of each element with logical AND operators. This is useful when composing a WHERE clause predicate that is based on a list of columns.
withVar	Takes an array and does not add additional formatting of the block. This helper is useful when filtering with Handlebars hash attributes.

Each of these default helpers can optionally include two Handlebars hash attributes: Value and NotValue. These attributes allow simple filtering of columns or tables based on column properties or based on custom variables.

Hash attributes (for helpers andList, orList, commaList and withVar	Description
Value="`string`"	Filters columns with a specific value.
NotValue="`string`"	Filters columns that do not have a specific value.

Custom Handlebars helpers

To create your own custom helpers:

1. In a JavaScript file, create the helpers and register them by including a Handlebars.registerHelper call in the file.
   Do not name your custom helpers file the same as the default helper file (`sqlhelpers.js`)
2. Add the file to the templateHelpers array in the project file.

Example

Hash attribute examples

The following example produces a list of column names that have a templateVariable named customVar:

```
{{commaList columns varName="customVar"}}
```

```
 {{columnName}}
{{/commaList}}
```

The following example selects only columns with a templateVariable customVar set to value 1234:

```
{{commaList columns varName="customVar" value="1234" }}
 {{columnName}}
 {{/commaList}}
```

The following example selects only columns with a template variable customVar that don't have the value 1234:

```
{{commaList columns varName="customVar" notValue="1234" }}
{{columnName}}
{{/commaList}}
```

Customer helpers example

The following code is an example of a customized Handlebars helpers file:

```
Handlebars.registerHelper( "spaceList", function( items, options ) {
    var ret = "";
    for( var i=0; i < items.length; i++ ) {
        var item = items[i];
        if( filterByOptions( item, options ) ) continue;
        if( i != 0 ) {
            ret += " ";
        }
        ret += options.fn( item );
    }
    return ret;
});
```

Template examples

The following example is a customized Handlebars template file that contains synchronization scripts:

```
{!-- This is a Handlebars template.  See http://handlebarsjs.com for
documentation. --}}
---------------------------------------------------------------
-- APPLICATION:    Data Synchronization
--
-- Synchronization scripts for: {{consDBTableName}}
--
-- AUTHOR:     Generated
--
-- Notes:
--------------------------------------------------------------
CALL ml_add_table_script (
    '{{scriptVersion}}',
    '{{consDBTableName}}',
    'download_cursor',
    'CALL {{scriptVersion}}_{{consDBTableName}}_DC (
        {ml s.last_table_download}, {ml s.username} )'
);
CALL ml_add_table_script (
    '{{scriptVersion}}',
    '{{consDBTableName}}',
    'download_delete_cursor',
    'CALL {{scriptVersion}}_{{consDBTableName}}_DDC (
        {ml s.last_table_download}, {ml s.username} )'
);
CALL ml_add_table_script (
    '{{scriptVersion}}',
    '{{consDBTableName}}',
    'upload_insert',
    'CALL {{scriptVersion}}_{{consDBTableName}}_UI (
     {{#commaList columns}}{ml r.{{columnName}} }{{/commaList}})'
);
CALL ml_add_table_script (
    '{{scriptVersion}}',
    '{{consDBTableName}}',
    'upload_delete',
    'CALL {{scriptVersion}}_{{consDBTableName}}_UD (
     {{#commaList primaryKeyColumns}}{ml r.{{columnName}} }{{/commaList}})'
);
CALL ml_add_table_script (
    '{{scriptVersion}}',
    '{{consDBTableName}}',
    'upload_update',
    'CALL {{scriptVersion}}_{{consDBTableName}}_UU (
     {{#commaList nonPrimaryKeyColumns}}{ml r.{{columnName}} }{{/commaList}},
     {{#commaList primaryKeyColumns}}{ml r.{{columnName}} }{{/commaList}}
     )'
);
```

The following example is of a customized Handlebars template that contains trigger definitions:

```
{{!-- This is a Handlebars template.  See http://handlebarsjs.com for
documentation. --}}
---------------------------------------------------------------
-- APPLICATION:    Data Synchronization
--
-- Trigger script for: {{consDBTableName}}
--
-- AUTHOR:     Generated
--
-- Notes:
--------------------------------------------------------------
CREATE OR REPLACE TRIGGER {{consDBTableOwner}}.TRIG_INSERT_{{consDBTableName}}
AFTER INSERT, UPDATE ON {{consDBTableName}}
REFERENCING NEW AS new_row
FOR EACH ROW
BEGIN
    INSERT INTO {{consDBTableName}}_SHADOW ({{primaryKeyColumnNames}},
        OPERATION, LAST_MODIFIED)
    ON EXISTING UPDATE
    VALUES ({{#commaList primaryKeyColumns}}new_row.{{columnName}} {{/
commaList}},
        'U', CURRENT_TIMESTAMP);
 END;
CREATE OR REPLACE TRIGGER {{consDBTableOwner}}.TRIG_DELETE_{{consDBTableName}}
AFTER DELETE ON {{consDBTableName}}
REFERENCING OLD AS old_row
FOR EACH ROW
BEGIN
   UPDATE {{consDBTableName}}_SHADOW as SHADOW
   SET OPERATION = 'D'
   WHERE {{#andList primaryKeyColumns}}SHADOW.{{columnName}} = old_row.
{{columnName}}
   {{/andList}}
END;
```

Related Information

http://handlebarsjs.com/

MobiLink Template Utility (mltemplate) [page 339]

Rules (mltemplate) [page 348]

MobiLink Template Utility (mltemplate) [page 339]

1.7.8 Output Files (mltemplate)

Organize the generated content from mltemplate by grouping the output by template and/or by rule.

By default, all generated output from mltemplate is saved to a single file. When all of the output is saved to one file, it can be difficult to compare the output of different generations because the file can be too large. A single file is also not useful when planning incremental changes, or when generating SQL for both consolidated and remote databases. In many cases it is better to generate your content to multiple output files.

Use a dedicated directory to store the output files by specifying a directory for the outDir field in the project file. The default value for outDir is null, which results in the output files being saved to the mltemplate working directory.

Customize the output files in the following ways:

Output file level	Description
Global output file	Change the name of the default output file by specifying a different value for the top-level outFile field in the project file. By default, all the generated content is saved to this SQL file unless you specify otherwise by using one of the methods listed below. The default value is `generatedScripts.txt`. The specified value can be overridden at generation time by providing more specific entries in the project file (below) or by specifying the -o option with *mltemplate gen*.

Per template output file	Group the output by the template used to generate it. Use the templateOutFiles object to specify the template file and its output file. For each template file named in the templateOutFiles object, an output file is created that contains all the output generated by running the template.
	
	For example, it could be useful when generating shadow table definitions for all the tables in your project to save them to one file.
	
	Use the following syntax to define the per template output files
	
	`"templateOutFiles": {`
	`{ "templateFileName1": "outputFile1" [, ...]},`

Per rule output file	Specify an output file for a rule by specifying the outFile field for the rule. For each rule that has an outFile field defined, an output file is created. This rule takes precedence over the other rules.

Example

The following example sets the name of default output file to mobiLinkScripts.sql:

```
...
 "outDir": null,
 "outFile" : "mobiLinkScripts.sql",
...
```

1.7.9 Tutorial: MobiLink Template System Utility (mltemplate)

Use the mltemplate to create a synchronization project that uses SQL Anywhere consolidated and client databases.

Prerequisites

The mltemplate sample, which is located in the *%SQLANYSAMP17%*`\ Mobilink\MLTemplate` directory. This tutorial uses portions of this sample, and it is recommended that you read its accompanying read me file.

You must have access to the SQL Anywhere 17 Demo database (demo.db).

You must have the SYS_AUTH_RESOURCE_ROLE to create the database objects in the consolidated and client databases.

(Optional.) The Apache Ant build tool. This tutorial uses Apache Ant to create the consolidated and client databases. Alternatively, you can use the instructions in the sample's read me to create the databases without using Apache Ant. Apache Ant is not supplied with SQL Anywhere, but it is available from http://ant.apache.org/ .

Context

The best way to learn how to use mltemplate is to study and experiment with an example project. This tutorial uses the mltemplate sample as the example project. Because the mltemplate sample is a complete project, not all lessons involve actions, some lessons discuss the objects in the sample.

The mltemplate sample is a synchronization project that uses the SQL Anywhere sample database as the consolidated database, and which creates client databases that contain the Products and Customers tables. The sample uses a simple synchronization logic in which all changes to these tables in the remote database are replicated to the consolidated database, and vice versa. The synchronization logic does not include conflict resolution, any form of data partitioning, or guarantee uniqueness of primary key values.

In this section:

Set up the directory structure for your synchronization project, and you use mltemplate to create your project file.

Lesson 2: Importing Database Schemas [page 366]
Import the database schema of two tables from the SQL Anywhere sample into your `myproject.js` project file.

Lesson 3: Learning About How Rules, Variables, and Handlebars Templates Interact [page 368]
This lesson describes customized Handlebars template provided with the mltemplate sample.

Lesson 4: Building Consolidated and Client Databases for the mltemplate Sample [page 369]
In this lesson, you build the consolidated and client databases for the mltemplate sample and simulate synchronizations between them.

Lesson 5: Cleaning up [page 371]
Remove the tutorials materials from your computer.

Related Information

The Project File (mltemplate) [page 344]
http://handlebarsjs.com/
MobiLink Template Utility (mltemplate) [page 339]

1.7.9.1 Lesson 1: Creating a Project File

Set up the directory structure for your synchronization project, and you use mltemplate to create your project file.

Prerequisites

You must have the roles and privileges listed at the beginning of this tutorial.

Context

The mltemplate sample contains a completed project file that we want to preserve. This lesson walks you through the first step of creating a different project file.

Procedure

1. Create a copy of the mltemplate sample to work with. Copy the *%SQLANYSAMP17%*`\ Mobilink \MLTemplate` directory, including its sub-directories and files, to a location where you have read and write access.

   For example copy the files to your `C:\` directory.

   The `MLTemplate` directory contains the following sub-directories:

   - The Helpers directory contains the customized Handlebars helper files.
   - The Templates directory contains the customized Handlebars templates files.

   The `MLTemplate` directory contains the project file, so it is your working directory. Always run mltemplate from this directory because mltemplate interprets relative paths as being relative to the directory from which it runs.
2. Create a project file. From the command prompt, run the following mltemplate command (all on one line) from the `MLTemplate` directory:

   ```
   mltemplate new myproject.json
   -t Template -js Helper/custom_helpers.js
   -s V1 -remType sqla -consType sqla
   ```

   This command creates a project file with following settings:

   - It sets the directory for the Handlebars template files to be the sample's customized template directory (`c:\MLTemplate\Template`). In the project file, this directory is added to the TemplateRoots array
     The files in `c:\MLTemplate\Template` were created by copying the default Handlebars templates from *%SQLANY17%*`\MobiLink\MLTemplate\Template`, renaming them (to avoid mltemplate using the default files), and then customizing them for this synchronization project.
     It adds the sample's customized Handlebars helper file (*%SQLANYSAMP17%*`\Mobilink \MLTemplate\Helper\custom_helpers.js`) to the list of helper files used by the project. In the project file, this file is added to the templateHelpers array, which also contains the default `sqlhelpers.js` file.
   - It specifies that the database type for both the consolidated and client databases is SQL Anywhere.
   - It names the version of the MobiLink synchronization scripts as V1.

   The project file is created and it contains the environmental information about the project, such as the database types of the client and consolidated databases, and the location of the Handlebars template directory and the Handlebars helpers file.
3. Open `myproject.json` in a text editor and replace the global templateVariables array (at the beginning of the project file) with the following array:

   ```
   "templateVariables": {
         "scriptVersion": "V1",
         "consDBTableOwner": "GROUPO",
         "projectName": "sqla_demo"
     },
   ```
4. In the project file, locate the default rule (named default) and replace the default Handlebars templates with the customized ones from the sample.

For SQL Anywhere consolidated databases, the default rule specifies two default template files: snapshot_download_scripts.hbs and upload_scripts.hbs. These default templates are located in the default template directory, *%SQLANY17%*`\MobiLink\MLTemplate\Templates`.

```
"templateRules": [
    {
      "ruleName": "default",
      "outFile": null,
      "tables": [],
      "templateVariables": {},
      "templateFiles": [
        "snapshot_download_scripts.hbs",
        "upload_scripts.hbs"
      ]
    },
```

The mltemplate sample uses only customized templates, so you must remove the default files and replace them with the customized ones from the sample.

Replace the references to the default template files in the default rule with the following customized Handlebars templates files

```
...
      "templateFiles": [
        "template/mobilink/cdb/sqla/basic/shadow_table.hbs",
        "template/mobilink/cdb/sqla/basic/trigger.hbs",
        "template/mobilink/cdb/sqla/basic/sync_script.hbs",
        "template/mobilink/cdb/sqla/basic/stored_procedure.hbs",
        "template/mobilink/cdb/sqla/basic/load.hbs"
      ]
...
```

mltemplate always searches its default templates directory before searching for any customized templates. Therefore never name a customized template file the same name as a default template file.

5. Replace the default rule conn_script with the following global rules:

```
{ "ruleName": "cdb_report",
      "tables": null,
      "templateVariables": { },
      "templateFiles": [
          "template/mobilink/cdb/sqla/test/report.hbs"
      ]
    },
    {
        "ruleName": "rdb_tables",
        "outFile": null,
        "tables": [
            "Products",
            "Customers"
        ],
        "templateVariables": {},
        "templateFiles": [
            "template/mobilink/rdb/sqla/table.hbs"
        ]
      },
      { "ruleName": "publication",
        "tables": null,
        "templateVariables": { },
        "templateFiles": [
            "template/mobilink/rdb/sqla/publication.hbs"
        ]
      },
      { "ruleName": "subscription",
        "tables": null,
        "templateVariables": {
            "ml_host": "localhost",
            "ml_port": "2439"
        },
        "templateFiles": [
            "template/mobilink/rdb/sqla/subscription.hbs"
        ]
      }
```

The templateRules array in `myproject.json` should resemble the following code:

```
...
  "templateRules": [
    {
      "ruleName": "default",
      "outFile": null,
      "tables": [
        "Products"
      ],
      "templateVariables": {},
      "templateFiles": [
          "template/mobilink/cdb/sqla/basic/shadow_table.hbs",
          "template/mobilink/cdb/sqla/basic/trigger.hbs",
          "template/mobilink/cdb/sqla/basic/sync_script.hbs",
          "template/mobilink/cdb/sqla/basic/stored_procedure.hbs",
          "template/mobilink/cdb/sqla/basic/load.hbs"
      ]
    },
    { "ruleName": "cdb_report",
      "tables": null,
      "templateVariables": { },
      "templateFiles": [
          "template/mobilink/cdb/sqla/test/report.hbs"
      ]
    },
    {
        "ruleName": "rdb_tables",
        "outFile": null,
        "tables": [
            "Products"
        ],
        "templateVariables": {},
        "templateFiles": [
            "template/mobilink/rdb/sqla/table.hbs"
        ]
      },
      { "ruleName": "publication",
        "tables": null,
        "templateVariables": { },
        "templateFiles": [
            "template/mobilink/rdb/sqla/publication.hbs"
        ]
      },
      { "ruleName": "subscription",
        "tables": null,
        "templateVariables": {
            "ml_host": "localhost",
            "ml_port": "2439"
        },
        "templateFiles": [
            "template/mobilink/rdb/sqla/subscription.hbs"
        ]
      },
      { "ruleName": "rdb_report",
        "tables": null,
        "templateVariables": { },
        "templateFiles": [
            "template/mobilink/rdb/sqla/test/report.hbs"
        ]
      },
      { "ruleName": "rdb_insert",
        "tables": null,
        "templateVariables": { },
        "templateFiles": [
            "template/mobilink/rdb/sqla/test/insert.hbs"
        ]
      }
  ],
  "syncTables": [
...
```

Next Steps

Proceed to the next lesson.

1.7.9.2 Lesson 2: Importing Database Schemas

Import the database schema of two tables from the SQL Anywhere sample into your `myproject.js` project file.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Procedure

1. Import the database schema of the SQL Anywhere sample database by running the following command (on one line):

   ```
   mltemplate import myproject.json
   -jdbc "jdbc:sqlanywhere:DSN=SQL Anywhere 17 Demo;Password=sql"
   -t Products -t Customers
   ```

   The table definition for the Products and Customers table along with the schema variables for these tables are added to the syncTables array in the `myproject.json` file.
2. Open `myproject.json` in a text editor, and verify that the SyncTables array contains the schema of the two tables.

Within each table is an array of columns. Each column object contains property fields for the remote schema as well as a cons field that contains property fields for the consolidated schema.

The following example shows the properties for the ID column of the Products table:

```
"syncTables": [
    {
      "tableName": "Products",
      "consDBTableName": "Products",
      "templateVariables": {},
      "columns": [
        {
          "columnName": "ID",
          "templateVariables": {},
          "baseDomain": "INTEGER",
          "domain": "INTEGER",
          "size": null,
          "scale": null,
          "nullable": 0,
          "ordinal": 1,
          "cons": {
            "columnName": "ID",
            "baseDomain": "INTEGER",
            "domain": "INTEGER",
            "size": null,
            "scale": null,
            "nullable": 0
          }
        },
...
```

You can verify the contents of your project file with the `project.json` file from the sample.

Next Steps

Proceed to the next lesson.

Related Information

MobiLink Template Utility (mltemplate) [page 339]

1.7.9.3 Lesson 3: Learning About How Rules, Variables, and Handlebars Templates Interact

This lesson describes customized Handlebars template provided with the mltemplate sample.

Context

The mltemplate sample contains customized Handlebars templates for both the consolidated and remote databases. Read the sample's readme.txt file for a description of each template file.

Handlebars templates can contain the following types of variables:

1. Schema variables, which are added to the project file when the database schema is imported. In the publication template, syncTables and tableName are examples of schema variables.
2. Custom variables, which are manually added to templateVariables objects in the project file. In the publication template, projectName is an example of a custom variable.
3. Customized Handlebars helpers. In the mltemplate sample, the customized Handlebars helpers are located in `MLTemplates\helper\custom_helpers.js`. In the publication template, commaList is an example of a customized helper.

Procedure

1. Open `myproject.json` in a text editor and find the publication rule.

   ```
   {
        "ruleName": "publication",
        "tables": null,
        "templateVariables": {},
        "templateFiles": [
          "mobilink/rdb/sqla/publication.hbs"
        ],
        "outFile": null
      },
   ```

   Because the tables field is set to null, the rule is a global rule, which means that it applies to all tables defined in the project file. The publication rule applies the Handlebars template file `publication.hbs` to the tables in the project file.
2. Open the `publication.hbs` in a text editor:

   The following code contains the contents of the publication template (`publication.hbs`).

   ```
   -- create a synchronization profile and other artifacts for the client
   -- Publication
   DROP PUBLICATION IF EXISTS  PUB_{{projectName}}
   GO
   CREATE PUBLICATION PUB_{{projectName}}
   (
       {{#commaList syncTables}}TABLE {{tableName}}{{/commaList}}
   )
   GO
   ```

3. Generate the output and save it to the default file, `generatedScripts.sql`, by running the following command:

```
mltemplate gen myproject.json
```

When mltemplate gen is run, each rule in the project file is run. The variables from the project file, the template files, and the helper files are collected and passed to the templates, so that they are available for use by the templates.

The following code contains the SQL output generated for the Products table from the Publication template:

```
DROP PUBLICATION IF EXISTS  PUB_sqla_demo
GO
CREATE PUBLICATION PUB_sqla_demo
(
    TABLE Products
)
GO
```

Next Steps

Proceed to the next lesson.

Related Information

Rules (mltemplate) [page 348]
Handlebars Templates and Helpers (mltemplate) [page 356]
MobiLink Template Utility (mltemplate) [page 339]

1.7.9.4 Lesson 4: Building Consolidated and Client Databases for the mltemplate Sample

In this lesson, you build the consolidated and client databases for the mltemplate sample and simulate synchronizations between them.

Prerequisites

You must have completed the previous lessons in this tutorial.

You must have the roles and privileges listed at the beginning of this tutorial.

Context

Use the Ant build tool and the `build.xml` file that is included with the mltemplate sample to create a consolidated and client database. Then, simulate a synchronization between the databases. The build.xml file instructs Ant to run mltemplate gen, so you do not need to run this yourself.

The readme for the mltemplate sample describes how to perform the same actions without using Ant.

Run Ant from the same directory that you used to run mltemplate.

This lesson uses the project file that comes with the mltemplate, and not the project file that you created.

Procedure

1. Create a remote database by running the following command:

   ```
   ant rdb
   ```

   A client database is created for a synchronization user named ML_USER_1. To create a remote database for another user, change the MLUSER environment variable and rerun the command.
2. Create the synchronization objects in the consolidated database by running the following command.

   ```
   ant cdb
   ```

   The default DBA user connects to the SQL Anywhere sample database and adds the required synchronization setup scripts and objects.
3. Start the MobiLink server running against the consolidated database.

   ```
   ant mobilink
   ```
4. Synchronize the client database..

   ```
   ant sync
   ```
5. Inspect the tables in the consolidated and client databases by running the following command:

   ```
   ant report
   ```

   The synchronization scripts in this sample are simplistic: they do no partitioning of the data and there is no provision to avoid conflicts if changes are made at remote databases.

Next Steps

Proceed to the next lesson.

Related Information

Rules (mltemplate) [page 348]
MobiLink Template Utility (mltemplate) [page 339]

1.7.9.5 Lesson 5: Cleaning up

Remove the tutorials materials from your computer.

Procedure

1. Close all instances of the any applications that you are running as a result of this tutorial:
2. Delete the directories that you created to contain the mltemplate sample files
3. Run the following command to erase the sample database and create a new copy of the sample database with its original objects and data:

   ```
   newdemo "%SQLANYSAMP17%\demo.db"
   ```

   When you are prompted, choose to erase any existing files.

   The tutorial materials are removed from your computer.

Results

You have completed the mltemplate tutorial.

Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.
About the icons:

- Links with the icon : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
  - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
  - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.
The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

www.sap.com/contactsap

THE BEST RUN SAP