iAnywhere®
A SYBASE COMPANY

MobiLink Clients

Part number: DC00193-01-0902-01
Last modified: October 2004

Contents

About This Manual

Subject

This manual describes MobiLink, a session-based relational-database synchronization system. MobiLink technology allows two-way replication and is well suited to mobile computing environments.

Audience

This manual is for users of Adaptive Server Anywhere and other relational database systems who wish to add synchronization or replication to their information systems.

Before you begin

☞ For a comparison of MobiLink with other synchronization and replication technologies, see “ Introducing Replication Technologies” [*Introducing SQL Anywhere Studio,* page 21].

SQL Anywhere Studio documentation

This book is part of the SQL Anywhere documentation set. This section describes the books in the documentation set and how you can use them.

The SQL Anywhere Studio documentation

The SQL Anywhere Studio documentation is available in a variety of forms: in an online form that combines all books in one large help file; as separate PDF files for each book; and as printed books that you can purchase. The documentation consists of the following books:

- **Introducing SQL Anywhere Studio** This book provides an overview of the SQL Anywhere Studio database management and synchronization technologies. It includes tutorials to introduce you to each of the pieces that make up SQL Anywhere Studio.
- **What's New in SQL Anywhere Studio** This book is for users of previous versions of the software. It lists new features in this and previous releases of the product and describes upgrade procedures.
- **Adaptive Server Anywhere Database Administration Guide** This book covers material related to running, managing, and configuring databases and database servers.
- **Adaptive Server Anywhere SQL User's Guide** This book describes how to design and create databases; how to import, export, and modify data; how to retrieve data; and how to build stored procedures and triggers.
- **Adaptive Server Anywhere SQL Reference Manual** This book provides a complete reference for the SQL language used by Adaptive Server Anywhere. It also describes the Adaptive Server Anywhere system tables and procedures.
- **Adaptive Server Anywhere Programming Guide** This book describes how to build and deploy database applications using the C, C++, and Java programming languages. Users of tools such as Visual Basic and PowerBuilder can use the programming interfaces provided by those tools. It also describes the Adaptive Server Anywhere ADO.NET data provider.
- **Adaptive Server Anywhere SNMP Extension Agent User's Guide** This book describes how to configure the Adaptive Server Anywhere SNMP Extension Agent for use with SNMP management applications to manage Adaptive Server Anywhere databases.
- **Adaptive Server Anywhere Error Messages** This book provides a complete listing of Adaptive Server Anywhere error messages together with diagnostic information.

- **SQL Anywhere Studio Security Guide** This book provides information about security features in Adaptive Server Anywhere databases. Adaptive Server Anywhere 7.0 was awarded a TCSEC (Trusted Computer System Evaluation Criteria) C2 security rating from the U.S. Government. This book may be of interest to those who wish to run the current version of Adaptive Server Anywhere in a manner equivalent to the C2-certified environment.
- **MobiLink Synchronization User’s Guide** This book describes how to use the MobiLink data synchronization system for mobile computing, which enables sharing of data between a single Oracle, Sybase, Microsoft or IBM database and many Adaptive Server Anywhere or UltraLite databases.
- **MobiLink Administration Guide** This book describes how to use the MobiLink data synchronization system for mobile computing, which enables sharing of data between a single Oracle, Sybase, Microsoft or IBM database and many Adaptive Server Anywhere or UltraLite databases.
- **MobiLink Clients** This book describes how to set up and synchronize Adaptive Server Anywhere and UltraLite remote databases.
- **MobiLink Tutorials** This book provides several tutorials that walk you through how to set up and run MobiLink applications.
- **QAnywhere User’s Guide** This manual describes MobiLink QAnywhere, a messaging platform that enables the development and deployment of messaging applications for mobile and wireless clients, as well as traditional desktop and laptop clients.
- **iAnywhere Solutions ODBC Drivers** This book describes how to set up ODBC drivers to access consolidated databases other than Adaptive Server Anywhere from the MobiLink synchronization server and from Adaptive Server Anywhere remote data access.
- **SQL Remote User’s Guide** This book describes all aspects of the SQL Remote data replication system for mobile computing, which enables sharing of data between a single Adaptive Server Anywhere or Adaptive Server Enterprise database and many Adaptive Server Anywhere databases using an indirect link such as e-mail or file transfer.
- **SQL Anywhere Studio Help** This book includes the context-sensitive help for Sybase Central, Interactive SQL, and other graphical tools. It is not included in the printed documentation set.

- **UltraLite Database User's Guide** This book is intended for all UltraLite developers. It introduces the UltraLite database system and provides information common to all UltraLite programming interfaces.
- **UltraLite Interface Guides** A separate book is provided for each UltraLite programming interface. Some of these interfaces are provided as UltraLite components for rapid application development, and others are provided as static interfaces for C, C++, and Java development.

In addition to this documentation set, PowerDesigner and InfoMaker include their own online documentation.

Documentation formats

SQL Anywhere Studio provides documentation in the following formats:

- **Online documentation** The online documentation contains the complete SQL Anywhere Studio documentation, including both the books and the context-sensitive help for SQL Anywhere tools. The online documentation is updated with each maintenance release of the product, and is the most complete and up-to-date source of documentation.

  To access the online documentation on Windows operating systems, choose Start ➤ Programs ➤ SQL Anywhere 9 ➤ Online Books. You can navigate the online documentation using the HTML Help table of contents, index, and search facility in the left pane, as well as using the links and menus in the right pane.

  To access the online documentation on UNIX operating systems, see the HTML documentation under your SQL Anywhere installation.
- **PDF books** The SQL Anywhere books are provided as a set of PDF files, viewable with Adobe Acrobat Reader.

  The PDF books are accessible from the online books, or from the Windows Start menu.
- **Printed books** The complete set of books is available from Sybase sales or from eShop, the Sybase online store, at *http://eshop.sybase.com/eshop/documentation*.

Documentation conventions

This section lists the typographic and graphical conventions used in this documentation.

Syntax conventions

The following conventions are used in the SQL syntax descriptions:

- **Keywords** All SQL keywords appear in upper case, like the words ALTER TABLE in the following example:

  ALTER TABLE [*owner*.]*table-name*

- **Placeholders** Items that must be replaced with appropriate identifiers or expressions are shown like the words *owner* and *table-name* in the following example:

  ALTER TABLE [*owner*.]*table-name*

- **Repeating items** Lists of repeating items are shown with an element of the list followed by an ellipsis (three dots), like *column-constraint* in the following example:

  ADD *column-definition* [*column-constraint*, . . .]

  One or more list elements are allowed. In this example, if more than one is specified, they must be separated by commas.

- **Optional portions** Optional portions of a statement are enclosed by square brackets.

  RELEASE SAVEPOINT [*savepoint-name*]

  These square brackets indicate that the *savepoint-name* is optional. The square brackets should not be typed.

- **Options** When none or only one of a list of items can be chosen, vertical bars separate the items and the list is enclosed in square brackets.

  [**ASC** | **DESC**]

  For example, you can choose one of ASC, DESC, or neither. The square brackets should not be typed.

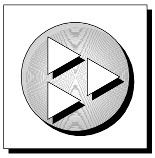
- **Alternatives** When precisely one of the options must be chosen, the alternatives are enclosed in curly braces and a bar is used to separate the options.

  [**QUOTES** { **ON** | **OFF** }]

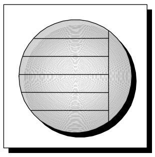
  If the QUOTES option is used, one of ON or OFF must be provided. The brackets and braces should not be typed.

Graphic icons The following icons are used in this documentation.

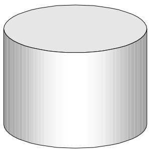
- A client application.



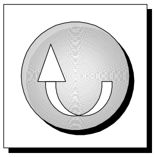
- A database server, such as Sybase Adaptive Server Anywhere.



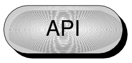
- A database. In some high-level diagrams, the icon may be used to represent both the database and the database server that manages it.



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



- A programming interface.



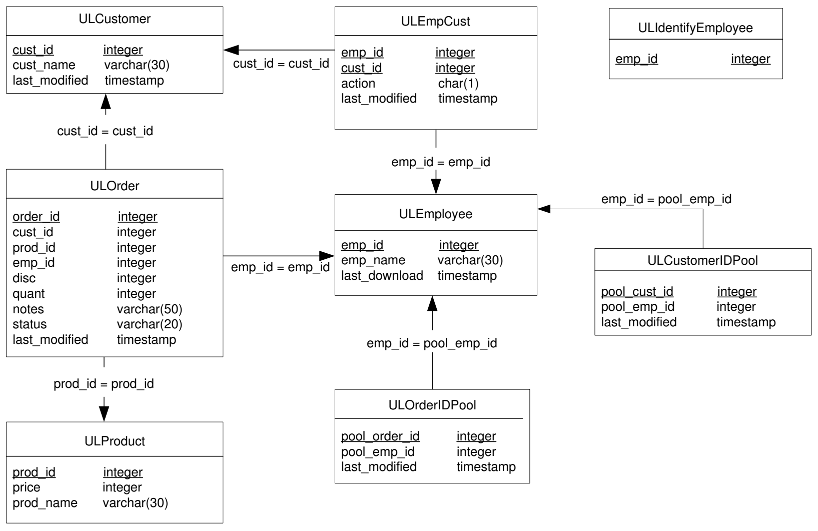
The CustDB sample database

Many of the examples in the MobiLink and UltraLite documentation use the UltraLite sample database.

The reference database for the UltraLite sample database is held in a file named *custdb.db*, and is located in the *Samples\UltraLite\CustDB* subdirectory of your SQL Anywhere directory. A complete application built on this database is also supplied.

The sample database is a sales-status database for a hardware supplier. It holds customer, product, and sales force information for the supplier.

The following figure shows the tables in the CustDB database and how they are related to each other.



Finding out more and providing feedback

Finding out more

Additional information and resources, including a code exchange, are available at the iAnywhere Developer Network at *http://www.ianywhere.com/developer/*.

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

When you write to one of these newsgroups, always provide detailed information about your problem, including the build number of your version of SQL Anywhere Studio. You can find this information by typing **dbeng9 -v** at a command prompt.

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

- sybase.public.sqlanywhere.general
- sybase.public.sqlanywhere.linux
- sybase.public.sqlanywhere.mobilink
- sybase.public.sqlanywhere.product_futures_discussion
- sybase.public.sqlanywhere.replication
- sybase.public.sqlanywhere.ultralite
- ianywhere.public.sqlanywhere.qanywhere

Newsgroup disclaimer

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

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

Feedback

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

You can e-mail comments and suggestions to the SQL Anywhere documentation team at iasdoc@ianywhere.com. Although we do not reply to e-mails sent to that address, we read all suggestions with interest.

In addition, you can provide feedback on the documentation and the software through the newsgroups listed above.

Part I

Introduction to MobiLink Clients

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

CHAPTER 1

Introducing MobiLink Clients

About this chapter

This chapter introduces you to MobiLink remote databases.

MobiLink supports two types of remote database: Adaptive Server Anywhere and UltraLite.

Contents

Adaptive Server Anywhere clients

Synchronization is initiated by running a command line utility called dbmlsync. This utility connects to the remote database and prepares the upload stream using information contained in the transaction log of the remote database. It then uses information stored in a synchronization publication and synchronization subscription to connect to the MobiLink synchronization server and exchange data.

☞ For more information about Adaptive Server Anywhere clients, see "Adaptive Server Anywhere Clients" on page 59.

☞ For details of dbmlsync command line options, see "Adaptive Server Anywhere Client Synchronization Parameters" on page 95.

UltraLite clients

Applications built with the UltraLite technology available in SQL Anywhere Studio are automatically MobiLink-enabled whenever the application includes a call to the appropriate MobiLink synchronization function. The UltraLite development tools included in SQL Anywhere Studio automatically include synchronization logic when you build your UltraLite application.

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.

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

The interface for HotSync is slightly different. Once synchronization is initiated from the application or from HotSync, the MobiLink synchronization server and the UltraLite runtime control the actions that occur during synchronization.

☞ For more information about UltraLite clients, see "UltraLite Clients" on page 277.

☞ For more information about initiating synchronization, see "UltraLite Synchronization Parameters" on page 315.

Specifying the communications protocol for clients

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

dbmlsrv9 -c "*connection-string***" -x** *protocol***(** *options* **)**

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

```
dbmlsrv9 -c "dsn=ASA 9.0 Sample" -x tcpip
```

You can configure your protocol using options of the form:

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

For example:

```
dbmlsrv9 -c "dsn=ASA 9.0 Sample" -x tcpip(
   host=localhost;port=2439)
```

☞ For more information about network protocols and protocol options, see "-x option" [*MobiLink Administration Guide*, page 214].

MobiLink users

You need to provide one unique MobiLink user name for each remote database in the MobiLink system. This name uniquely identifies each MobiLink remote database.

The ml_user MobiLink system table, located in the consolidated database, holds a list of MobiLink user names. The synchronization state of each user is recorded in the commit_state column or the progress column. This information ensures proper recovery if synchronization is interrupted.

☞ For more information about MobiLink users, see

- ♦ "About MobiLink users" on page 10
- ♦ "Creating MobiLink users" on page 10
- ♦ "ml_user" [*MobiLink Administration Guide*, page 531]

CHAPTER 2

Authenticating MobiLink Users

About this chapter

A MobiLink user is a name that uniquely identifies a remote database for synchronization.

This chapter describes how to manage MobiLink users, including the mechanisms provided to manage and authenticate their passwords.

☞ For more information about security options, see “MobiLink Transport-Layer Security” [*MobiLink Administration Guide,* page 165].

Contents

About MobiLink users

A **MobiLink user**, also called a **synchronization user**, is a name assigned to a remote database. Each MobiLink user name must be unique within the synchronization system.

MobiLink user names and passwords are not the same as database user names and passwords. MobiLink user names and passwords are used to uniquely identify remote databases, and optionally to authenticate the connection from the remote database to the MobiLink synchronization server.

You can also use user names to control the behavior of the synchronization server. You do so using the user name in synchronization scripts. For example, you can send remote databases different rows based on their user name.

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

Naming conventions

In large-scale deployments, you might find it useful to adopt a naming convention for your MobiLink user names. For example, you could adopt a naming convention whereby MobiLink names were created as *user:application*, where *user* identifies a person and *application* identifies a specific application, such as `EmpA01:HRApp`. Or your convention could be *user:application:device*, where *device* identifies a remote device.

UltraLite user authentication

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

☞ For more information about UltraLite user authentication, see "User authentication in UltraLite" [*UltraLite Database User's Guide*, page 40].

Creating MobiLink users

Warning
A MobiLink user name uniquely identifies a remote database. Therefore, two different remote databases must have different MobiLink user names.

Create users in the remote database

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

- For Adaptive Server Anywhere remotes, use Sybase Central or the CREATE SYNCHRONIZATION USER statement.

  ☞ For more information, see "Adding MobiLink users to a remote database" on page 71.

- For UltraLite remotes, you can use the user_name field of the ul_synch_info structure. In Java, use the SetUserName() method of the ULSynchInfo class before synchronizing.

  ☞ For more information, see "User Name synchronization parameter" on page 338 and "Password synchronization parameter" on page 324.

Add MobiLink user names to the consolidated database

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

- Use the dbmluser utility.

  ☞ For more information, see "MobiLink user authentication utility" [*MobiLink Administration Guide,* page 492].

- Use Sybase Central.

- Implement a script for the authenticate_user or authenticate_user_hashed events. When either of these scripts are invoked, the MobiLink synchronization server automatically adds users that successfully authenticate.

- Specify the -zu+ command line option with dbmlsrv9. In this case, any existing MobiLink users that have not been added to the consolidated database are added when they first synchronize.

  ☞ For more information, see "-zu option" [*MobiLink Administration Guide,* page 222].

Sharing MobiLink user names

MobiLink user names must be unique within a synchronization system. If you want two or more remote databases (UltraLite or Adaptive Server Anywhere) to share the same MobiLink user name, you can create a MobiLink user name that is a base name with a unique suffix.

A typical use of this technique is for a person who wants to have several remote databases. Each remote database must have a unique MobiLink user name, but they can share the same base name.

Example

The following example creates MobiLink user names that are 102 followed by a colon and a universally unique ID.

```
BEGIN
EXECUTE IMMEDIATE 'CREATE SYNCHRONIZATION USER "102' + ':' +
NEWID() + '"';
END;
```

This creates a MobiLink user name such as 102:b23fdbed-bead-418a-9d53-917e774c2f4f.

You still need MobiLink to provide the user name to each of the MobiLink scripts. To do this, you can use a MobiLink event called modify_user. It takes the MobiLink user as input and allows you to modify it. The modified value is what is passed to all the download events. For example,

```
CALL sp_ML_modify_user( ? )
```

The result is that the following download_cursor is based on the value of 102, not 102:b23fdbed-bead-418a-9d53-917e774c2f4f.

```
Select emp_id, emp_name
From ULEmployee
Where last_modified >= ?
 And emp_id = ?
```

Here is the procedure written using Adaptive Server Anywhere syntax. This can easily be converted for other RDBMSs.

```
CREATE PROCEDURE sp_ML_modify_user( INOUT @ml_user_name
         VARCHAR(255) )
BEGIN
    DECLARE @colon_at INT;
    SET @colon_at = LOCATE( @ml_user_name, ':' );
    IF( @colon_at > 0 ) THEN
        -- Message statements are displayed in the minimized
         engine
        -- window, this is useful for debugging

        MESSAGE 'UUID: ' +
                RIGHT( @ml_user_name,
                    (LENGTH(@ml_user_name)-@colon_at) );
        SET @ml_user_name = LEFT( @ml_user_name, (@colon_at-1)
         );
        MESSAGE 'New MobiLink User: ' + @ml_user_name;
    ELSE
        MESSAGE 'No change to MobiLink User: ' + @ml_user_name;
    END IF;
END;
```

Choosing a user authentication mechanism

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

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

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

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

  ☞ For more information about custom user authentication mechanisms, see "Custom user authentication" on page 21.

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

- ♦ "MobiLink Transport-Layer Security" [*MobiLink Administration Guide*, page 165]
- ♦ "Encrypting UltraLite databases" [*UltraLite Database User's Guide*, page 36]
- ♦ "Keeping Your Data Secure" [*SQL Anywhere Studio Security Guide*, page 3]

User authentication architecture

The MobiLink user authentication system relies on user names and passwords. You can choose either to let the MobiLink synchronization 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 synchronization server cannot read the ml_user table and reconstruct the original form of the password. You add user names and passwords to the consolidated database using Sybase Central, using the dbmluser utility, or by specifying -zu+ when you start the MobiLink synchronization server.

☞ For more information, see "Creating MobiLink users" on page 10.

When a MobiLink client connects to a MobiLink synchronization 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 synchronization 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.

☞ For more information, see "Custom user authentication" on page 21.

The authentication process

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

1. You initiate a synchronization request using a MobiLink user name, and optionally a password and new password. The MobiLink synchronization server starts a new transaction and triggers the begin_connection event.

2. MobiLink verifies that the MobiLink user name you provided is not currently synchronizing.

3. If you have defined an authenticate_user script, then the following occurs:
   a. The authenticate_user script is called and sets the auth_status field with the MobiLink user name you provided and optionally the password and new password.
   b. If the authenticate_user script throws an exception, the synchronization process stops.
   c. If the authenticate_user script returns a SQL statement, the statement is executed. The SQL statement must be a call to a stored procedure taking 2 to 4 arguments. The auth_status value that was set in step 3.a is passed as the first parameter and may be updated by the stored procedure.
4. If an error occurred, authentication stops with a failure.
5. Otherwise, if an authenticate_user_hashed script exists, then the following occurs:
   a. If a password was provided, a hashed value is calculated for it. If a new password was provided, a hashed value is calculated for it.
   b. The authenticate_user_hashed script is called with the current value of auth_status and the hashed passwords. The behavior is identical to steps 3.b and 3.c, except that the authenticate_user_hashed script can overwrite the value of auth_status with a larger (more severe) value if it was set by step 3.a. If the authenticate_user script was not defined, then the auth_status value that is set by authenticate_user_hashed is used no matter what.
6. The MobiLink synchronization server searches the ml_user table for the MobiLink user name you provided.
   a. If either of the custom scripts authenticate_user or authenticate_user_hashed was called but the MobiLink user name you provided is not in the ml_user table and the auth_status is valid (1000 or 2000), the MobiLink user name is added to the MobiLink system table ml_user. If auth_status is not valid, ml_user is not updated and an error occurs.
   b. If the custom scripts were not called and the MobiLink user name you provided is not in the ml_user table, the MobiLink user name you provided is added to ml_user if you started the MobiLink synchronization server with the -zu option. Otherwise, an error occurs and auth_status is set accordingly.
   c. If the custom scripts were called and the MobiLink user name you provided is in the ml_user table, nothing happens.

   d. If the custom scripts were not called and the MobiLink user name you provided is in the ml_user table, the password is checked against the value in the ml_user table and auth_status is set accordingly.

7. The MobiLink synchronization server attempts to authenticate the MobiLink user name you provided using the password you provided.
8. If neither of the scripts authenticate_user or authenticate_user_hashed was called and you provided a new password, the password is changed to the one you provided.
9. If you have defined an authenticate_parameters script and the auth_status is valid (1000 or 2000), then the following occurs:
   a. The parameters are passed to the authenticate_parameters script.
   b. If the authenticate_parameters script returns an auth_status value greater than the current auth_status, the new auth_status overwrites the old value.
10. If auth_status is not valid, the synchronization is aborted.
11. 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.
12. The MobiLink synchronization server commits or rolls back the synchronization as appropriate, and continues with the synchronization.

Providing initial passwords for users

The password for each user is stored along with the user name in the ml_user table. You can provide initial passwords from Sybase Central, or using the dbmluser command line utility.

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

If you create a user with no password, then MobiLink performs no user authentication for that user: they can connect and synchronize without supplying a password.

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

1. Connect to the consolidated database from Sybase Central using the MobiLink plug-in.
2. Open the Users folder.
3. Double-click Add User. The Add User wizard appears.
4. Supply a user name and an optional password.
5. Click Finish to complete the task.

❖ To provide initial MobiLink passwords (command line)

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

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

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

   ☞ For information about dbmluser, see “MobiLink user authentication utility” [*MobiLink Administration Guide*, page 492].

Synchronizations from new users

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

Setting the -zu+ option when you start the MobiLink synchronization server allows the MobiLink synchronization server to automatically add new user names to the ml_user table according to the following rules.

In effect, this option permits new users to create their own MobiLink accounts, easing administration of new users. This arrangement can be convenient when the server and clients all operate within a firewall.

If a MobiLink client synchronizes with a user name that is not in the current ml_user table, MobiLink, by default, takes the following actions:

- **New user, no password** If the user supplied no password, then by default the user name is added to the ml_user table with a NULL password. This behavior provides compatibility with earlier releases of MobiLink that did not allow user authentication.

  ☞ For more information, see "-zu option" [*MobiLink Administration Guide,* page 222].

- **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.

- **New user, new password** A new user may provide information in the new password field, instead of or as well as in the password field. In either case, the new password setting overrides the password setting, and the new user is added to the MobiLink system using the new password.

Preventing synchronization by unknown users

You can change the default behavior by starting the MobiLink synchronization server using the -zu option. In this case, the MobiLink synchronization server rejects any attempt to synchronize from a user name that is not present in the ml_user table.

This setting provides two benefits. First, it reduces the risk of unauthorized access to the MobiLink synchronization server. Second, it prevents authorized users from accidentally connecting using an incorrect or misspelled user name. Such accidents should be avoided because they can cause the MobiLink system to behave in unpredictable ways.

Prompting end users to enter passwords

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

❖ To prompt your end users to enter their MobiLink passwords

1. The mechanism for supplying the user name and password is different for UltraLite and Adaptive Server Anywhere clients.

    ♦ **UltraLite** When synchronizing, the UltraLite client must supply a valid value in the password field of the synchronization structure (C/C++) or object (Java). 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.

    ☞ For more information, see “UltraLite Synchronization Parameters” on page 315.

    ♦ **Adaptive Server Anywhere** You can supply a valid password on the dbmlsync command line. However, if you do not do so, you are prompted for one in the dbmlsync connection dialog. The latter method is more secure because command lines are visible to other processes running on the same computer.

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

    ☞ For more information, see “-c option” on page 102.

Changing passwords

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

❖ To prompt your end users to enter MobiLink passwords

1. The mechanism for supplying the user name and password is different for UltraLite and Adaptive Server Anywhere clients.
   - ♦ **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.

     ☞ For more information, see "Password synchronization parameter" on page 324 and "New Password synchronization parameter" on page 322.
   - ♦ **Adaptive Server Anywhere** Supply a valid existing password together with the new password on the dbmlsync command line, or in the dbmlsync connection dialog if you do not supply command line parameters.

     ☞ For more information, see "-mp option" on page 141 and "-mn option" on page 141.

The new password is not verified until the next synchronization attempt. For the dbmlsync utility, or if you prompt at synchronization time in an UltraLite application, this attempt is almost immediate.

☞ An initial password can be set in the consolidated server or on the first synchronization attempt. For more information, see "Providing initial passwords for users" on page 17 and "Synchronizations from new users" on page 18.

Once a password is assigned, you cannot reset the password to NULL from the client side.

Custom user authentication

You can choose to use a user authentication mechanism other than the built-in MobiLink mechanism. Reasons for using a custom user authentication mechanism include integration with existing DBMS user authentication schemes or external authentication mechanisms; or supplying 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:

♦ dbmlsrv9 -zu option

♦ authenticate_user script

♦ authenticate_parameters script

The dbmlsrv9 -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 script and authenticate_parameters script both override the default MobiLink user authentication mechanism. Any user that successfully authenticates is automatically added to the ml_user table

There are several predefined scripts for the authenticate_user event that are installed with MobiLink. These make it easier for you to authenticate using LDAP, POP3, and IMAP servers. For more information, see "Authenticating to external servers" on page 22.

Use authenticate_user 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.

For more information, see:

♦ "-zu option" [*MobiLink Administration Guide,* page 222]

♦ "authenticate_user connection event" [*MobiLink Administration Guide,* page 336]

♦ "authenticate_parameters connection event" [*MobiLink Administration Guide,* page 334]

Java and .NET user authentication

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

A simple sample is included in the directory *Samples\MobiLink\JavaAuthentication*. The sample code in *Samples\MobiLink\JavaAuthentication\CustEmpScripts.java* implements a simple user authentication system. On the first synchronization, a MobiLink user name is added to the login_added table. On subsequent synchronizations, a row is added to the login_audit table. In this sample, there is no test before adding a user ID to the login_added table.

For a .NET sample that explains user authentication, see ".NET synchronization example" [*MobiLink Administration Guide,* page 300].

SQL user authentication

A typical authenticate_user SQL script is a call to a stored procedure that uses the parameters auth_status, ml_username, user_password, and user_new_password. The order of the parameters in the call must match this. For example, in an Adaptive Server Anywhere consolidated database, the format is:

```
call my_authentication( ?, ?, ?, ? )
```

where the first argument is the authentication code, and so on. The authentication code is an integer type, and the other parameters are VARCHAR(128).

A Transact-SQL format is:

```
execute ? = my_authentication( ?, ?, ? )
```

where the authentication code is the parameter on the left hand side.

☞ For more information, see "authenticate_user connection event" [*MobiLink Administration Guide,* page 336].

Authenticating to external servers

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

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

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

If your MobiLink user names map directly to your user IDs

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

❖ To use predefined scripts directly in authenticate_user

1. Add the predefined Java synchronization script to the ml_scripts MobiLink system table. You can do this using a stored procedure or in Sybase Central.
   - To use the ml_add_java_connection_script stored procedure, type the following at a command prompt:

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

     where *MyVersion* is the name of a script version, and *ServerType* is **LDAP**, **POP3**, or **IMAP**.
   - To use the Add Connection Script wizard in Sybase Central, choose authenticate_user as the script type, and enter the following in the Code Editor:

     ```
       ianywhere.ml.authentication.ServerType.authenticate
     ```

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

2. Add properties for this authentication server.

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

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

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

☞ For more information, see "External authenticator properties" on page 25.

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

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

☞ For more information about writing Java classes, see "Writing Synchronization Scripts in Java" [*MobiLink Administration Guide*, page 255].

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

```
package com.mycompany.mycode;

import ianywhere.ml.authentication.*;
import ianywhere.ml.script.*;

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

    public MLEvents( DBConnectionContext context )
    {
        _context = context;
        _pop3 = new POP3( context );
    }
    public void authenticateUser(
      InOutInteger status,
      String userID,
      String password,
      String newPassword )
    {
        String realUserID = extractUserID( userID );
        _pop3.authenticate( status, realUserID, password,
         newPassword );
    }

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

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

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

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

External authenticator properties

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

POP3 authenticator

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

☞ For more information, see *http://java.sun.-com/products/javamail/javadocs/com/sun/mail/pop3/package-summary.html*

IMAP authenticator

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

☞ For more information, see *http://java.sun.com/products/javamail/javadocs/com/sun/mail/imap/package-summary.html*.

LDAP authenticator

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

☞ For more information, see the JNDI documentation.

CHAPTER 3

Utilities

About this chapter

This chapter describes MobiLink client utilities.

☞ For information about MobiLink synchronization server utilities, see "Utilities" [*MobiLink Administration Guide,* page 489].

☞ For information about other Adaptive Server Anywhere utilities, see "Database Administration Utilities" [*ASA Database Administration Guide,* page 493].

Contents

ActiveSync provider installation utility

Installs a MobiLink provider for ActiveSync, or registers and installs UltraLite applications on Windows CE devices.

Syntax

dbasinst [*options*] [[*src*] *dst name class* [*args*]]

Options	Description
-d	Disable the application on creation.
-k *path*	Specify the location of the desktop provider *dbasdesk.dll*.
-l *filename*	Write the activity log to the specified file. If no path or a relative path is given for the filename, the log file will be in or relative to the ActiveSync install directory.
-n	Register the application but do not copy it to the device.
-u	Uninstall the MobiLink ActiveSync provider.
-v *path*	Specify the location of the device provider *dbasdev.dll*.

Args	Description
src	The source filename and path for an application.
dst	The destination filename and path for an application.
name	The name of the application.
class	The registered Windows class name of the application.
args	Command line arguments to use when ActiveSync starts the application.

Description

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

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

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

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

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

- ♦ Adaptive Server Anywhere: "Installing the MobiLink provider for ActiveSync" on page 84
- ♦ UltraLite: "Installing the MobiLink provider for ActiveSync" on page 310

Options

-d By default, an application registered by dbasinst is enabled, meaning that it is automatically synchronized when ActiveSync begins a synchronization. With the -d option, the application is still registered, but it is unchecked in the ActiveSync MobiLink settings dialog.

-l The ActiveSync provider logs its activities to the specified file. If no path or a relative path is given for the filename, the log file is in or relative to the ActiveSync install directory.

-k The path to the desktop provider *dbasdesk.dll*. By default the file is looked for in the *win32* subdirectory of your SQL Anywhere directory. End users (who generally do not have the full SQL Anywhere install) may need to specify -k when installing the MobiLink ActiveSync provider.

-n In addition to installing the MobiLink ActiveSync provider, register an application but do not copy it to the device. This is appropriate if the application includes more than one file (for example, if it is compiled to use the UltraLite runtime library DLL rather than a static library) or if you have an alternative method of copying the application to the device.

-u Unregister all applications that have been registered for use with the MobiLink ActiveSync provider and uninstall the MobiLink ActiveSync provider. No files are deleted from the desktop machine or the device by this operation. If the device is not connected to the desktop, an error is reported.

-v The path to the device provider *dbasdev.dll*. By default the file is looked for in a platform-specific directory under the *CE* subdirectory of your SQL Anywhere directory. End users (who generally do not have the full SQL Anywhere install) may need to specify -v when installing the MobiLink ActiveSync provider.

Arguments

src The source filename and path for copying an application to the device. Supply this parameter only if you are registering an application and copying it to the device: do not supply the parameter if you use the -n option.

dst The destination filename and path on the device for an application.

name The application name. This is the name by which ActiveSync refers to the application.

class The registered Windows class name for the application.

args Any command line arguments to be used when ActiveSync starts the application.

Examples

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

```
dbasinst
```

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

```
dbasinst -u
```

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

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

See also

- "Using ActiveSync synchronization" on page 82
- "UltraLite Synchronization Parameters" on page 315

MobiLink client database extraction utility (deprecated)

Creates an Adaptive Server Anywhere client database using another Adaptive Server Anywhere database as a template. This utility is deprecated. For an alternative way to create client databases, see "Creating a remote database" on page 60.

Syntax

mlxtract [*additional-options*] *directory site-name*

Option	Description
-ac *"keyword=value; . . . "*	Connect to the database specified in the connect string to do the reload.
-al *filename*	Log file name for this new database.
-an *filename*	Creates a database file with the same settings as the database being unloaded and automatically reloads it.
-c *"keyword=value; . . . "*	Supply database connection parameters.
-id	Extract schema definition and data.
-it	Extract triggers.
-j *count*	Iteration count for view-creation statements.
-l *level*	Perform all extraction operations at specified isolation level.
-o *file*	Output messages to file.
-p *character*	Escape character.
-q	Operate quietly: do not print messages or show windows.
-r *file*	Specify name of generated reload Interactive SQL command file (default "*reload.sql*").
-s7	Use Adaptive Server Anywhere version 7 syntax for creating synchronization definitions.
-u	Unordered data.
-v	Verbose messages.
-x	Use external table loads.

Option	Description
-xh	Exclude procedure hooks.
-xf	Exclude foreign keys.
-xp	Exclude stored procedures.
-xv	Exclude views.
-y	Overwrite command file without confirmation.
directory	The directory to which the files are written. This option is not needed if you use `-an` or `-ac`.
site-name	Specify which client database to generate.

Description

mlxtract is the MobiLink extraction utility for Adaptive Server Anywhere client databases. It is run against an Adaptive Server Anywhere reference database and creates a new client database or a command file for an Adaptive Server Anywhere client database, depending on the chosen options.

The command line extraction utility creates a command file and a set of associated data files. The command file can be run against a newly initialized Adaptive Server Anywhere database to create the database objects and load the data for the client database.

By default, the command file is named *reload.sql*.

Options

Reload the data to an existing database (-ac) You can combine the operation of extracting a database and reloading the results into an existing database using this option.

For example, the following command (which should be entered all on one line) loads a copy of the data for the field_user subscriber into an existing database file named *newdemo.db*:

```
mlxtract -c "uid=DBA;pwd=SQL;dbf=asademo.db" -ac
         "uid=DBA;pwd=SQL;dbf=newdemo.db" field_user
```

If you use this option, no copy of the data is created on disk, so you do not specify an unload directory on the command line. This provides greater security for your data, but at some cost for performance.

Reload the data to a new database (-an) You can combine the operations of extracting a database, creating a new database, and loading the data using this option.

For example, the following command (which should be entered all on one line) creates a new database file named *asacopy.db* and copies the schema and data for the field_user subscriber of *asademo.db* into it:

```
mlxtract -c "uid=DBA;pwd=SQL;dbf=asademo.db" -an asacopy.db
         field_user
```

If you use this option, no copy of the data is created on disk, so you do not specify an unload directory on the command line. This provides greater security for your data, but at some cost for performance.

Connection parameters (-c) A set of connection parameters, in a string.

- **mlxtract connection parameters** The user ID should have DBA authority to ensure that the user has permissions on all the tables in the database.

  For example, the following statement (which should be typed on one line) extracts a database for MobiLink user name joe_remote from the ASADemo database running on the sample_server server, connecting as user ID DBA with password SQL. The data is unloaded into the *c:\extract* directory.

  ```
  mlxtract -c "eng=sample_server;dbn=sademo;
  uid=DBA;pwd=SQL" c:\extract joe_remote
  ```

Extract both schema definition and data (-id) By default, only the schema is extracted. Such a database can be initialized with data upon the first connection to a MobiLink synchronization server. This option provides the option of extracting the initial set of data from the reference database.

Extract triggers (-it) By default, triggers are not extracted. This option provides causes triggers present in the reference database to be extracted.

Iteration count for views (-j) If there are nested views in the consolidated database, this option specifies the maximum number of iterations to use when extracting the views.

Perform extraction at a specified isolation level (-l) The default setting is an isolation level of zero. If you are extracting a database from an active server, you should run it at isolation level 3 to ensure that data in the extracted database is consistent with data on the server. Increasing the isolation level may result in large numbers of locks being used by the extraction utility, and may restrict database use by other users.

Output messages to file (-o) Outputs the messages from the extraction process to a file for later review.

Escape character (-p) The default escape character (\) can be replaced by another character using this option.

Operate quietly (-q) Display no messages except errors.

Reload filename (-r) The default name for the reload command file is *reload.sql* in the current directory You can specify a different file name with this option.

Use ASA v7 syntax (-s7) This option is useful when you are using an Adaptive Server Anywhere version 8 or higher consolidated database along with Adaptive Server Anywhere version 7 remote databases. For example, create a version 9 consolidated database, extract the remote databases using the -s7 option, and deploy the reload.sql files to the remote.

Output the data unordered (-u) By default the data in each table is ordered by primary key. Unloads are quicker with the -u option, but loading the data into the client database is slower.

Verbose mode (-v) The name of the table being unloaded and the number of rows unloaded are displayed. The SELECT statement used is also displayed.

Use external loads (-x) In the reload script, the default is to use the LOAD TABLE statement to load the data into the database. If you choose to use external loads, the Interactive SQL INPUT statement is used instead. The LOAD TABLE statement is faster than INPUT.

INPUT takes the path of the data files relative to the client, while LOAD TABLE takes the path relative to the server.

Exclude foreign key definitions (-xf) You can use this if the client database contains a subset of the consolidated database schema, and some foreign key references are not present in the client database.

Exclude stored procedure (-xp) Do not extract stored procedures from the database.

Exclude views (-xv) Do not extract views from the database.

Operate without confirming actions (-y) Without this option, you are prompted to confirm the replacement of an existing reloadcommand file.

CHAPTER 4

MobiLink Client Network Protocol Options

About this chapter

This chapter describes the network protocol options that you use to connect to the MobiLink synchronization server from a remote database.

Contents

Protocol options

This chapter describes the network protocol options you can use when connecting a MobiLink client to the MobiLink synchronization server.

For information about how to use these options with dbmlsync, see “CommunicationAddress (adr) extended option” on page 106.

For information about how to use these options with UltraLite, see:

- “TCP/IP protocol options” on page 345
- “HTTP protocol options” on page 346
- “HTTPS protocol options” on page 347
- “UlSecureRSASocketStream synchronization parameters” on page 349
- “UlSecureSocketStream synchronization parameters” on page 350

For information about how to set connection options for the MobiLink synchronization server, see “-x option” [*MobiLink Administration Guide,* page 214].

buffer_size

Function	Specify the maximum HTTP body size for a fixed content length message, in bytes.
Syntax	**buffer_size=***number*
Protocols	♦ dbmlsync - HTTP, HTTPS, HTTPS_FIPS ♦ UltraLite - HTTP, HTTPS
Default	♦ UltraLite - 1024 ♦ dbmlsync on PocketPC - 1024 ♦ dbmlsync on other devices - 64 000
Remarks	In general, the larger the HTTP body size, the fewer the number of HTTP request-response cycles, but the more memory required for buffering the body.

HTTPS_FIPS is available on Windows only.

☞ For information about how to set network protocol options with dbmlsync, see “CommunicationAddress (adr) extended option” on page 106.

☞ For information about how to set network protocol options with UltraLite, see “Network protocol options for UltraLite synchronization clients” on page 341.

certificate_company

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

Separately licensable option required
Transport-layer security requires that you obtain a separately-licensable SQL Anywhere Studio security option and is subject to export regulations.

☞ To order this component, see “Separately-licensable components” [*Introducing SQL Anywhere Studio,* page 5].

Syntax

certificate_company=*organization*

Protocols

- ♦ dbmlsync - TCP/IP (via the security option), HTTPS, HTTPS_FIPS
- ♦ UltraLite - TCP/IP (via security parms), HTTPS, UlSecureRSASocketStream, UlSecureSocketStream

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 synchronization server for your own and accidentally send it sensitive information. This option specifies a further level of verification, that the Organization field in the identity portion of the certificate also matches a value you specify.

For TCP/IP, set this option as part of the security option. For more information, see “security” on page 50.

HTTPS_FIPS is available on Windows only.

☞ For information about how to set network protocol options with dbmlsync, see “CommunicationAddress (adr) extended option” on page 106.

☞ For information about how to set network protocol options with UltraLite, see “Network protocol options for UltraLite synchronization clients” on page 341.

See also

- ♦ “MobiLink Transport-Layer Security” [*MobiLink Administration Guide,* page 165]
- ♦ “Verifying certificate fields” [*MobiLink Administration Guide,* page 180]
- ♦ “-x option” [*MobiLink Administration Guide,* page 214]
- ♦ “security” on page 50
- ♦ “trusted_certificates” on page 53
- ♦ “certificate_name” on page 38

♦ “certificate_unit” on page 40

Example

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

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

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

With UltraLite clients, the precise syntax depends upon the interface used to build the application. The following fragment of C code accomplishes the same task when developing the UltraLite application using embedded SQL in C or C++:

```
ul_synch_info info;
info.stream = ULSocketStream();
info.stream_parms = TEXT("port=9999");
info.security_stream = ULSecureRSATLSStream();
info.security_parms =
   UL_TEXT ( "certificate_company=Sybase, Inc." )
   UL_TEXT ( ";" )
   UL_TEXT ( "certificate_unit=iAnywhere" )
   UL_TEXT ( ";" )
   UL_TEXT ( "certificate_name=sample" );
. . .
ULSynchronize( &info );
```

certificate_name

Function

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

Separately licensable option required
Transport-layer security requires that you obtain a separately-licensable SQL Anywhere Studio security option and is subject to export regulations.

☞ To order this component, see “Separately-licensable components” [*Introducing SQL Anywhere Studio*, page 5].

Syntax

certificate_name=*common-name*

Protocols

- dbmlsync - TCP/IP (via the security option), HTTPS, HTTPS_FIPS
- UltraLite - TCP/IP (via the security parameters), HTTPS, UlSecureRSASocketStream, UlSecureSocketStream

Default

None

Remarks

For TCP/IP, set this option as part of the security parameter. For more information, see "security" on page 50.

HTTPS_FIPS is available on Windows only.

☞ For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 106.

☞ For information about how to set network protocol options with UltraLite, see "Network protocol options for UltraLite synchronization clients" on page 341.

See also

- "MobiLink Transport-Layer Security" [*MobiLink Administration Guide,* page 165]
- "Verifying certificate fields" [*MobiLink Administration Guide,* page 180]
- "-x option" [*MobiLink Administration Guide,* page 214]
- "security" on page 50
- "trusted_certificates" on page 53
- "certificate_company" on page 37
- "certificate_unit" on page 40

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.

```
dbmlsrv9
   -c "dsn=asa90sample;uid=DBA;pwd=SQL"
   -x https(
     port=9999;
     certificate=c:\asa90\win32\rsaserver.crt;
     certificate_password=test)
```

On an Adaptive Server Anywhere client, the implementation is:

```
dbmlsync
   -c "dsn=mydb;uid=DBA;pwd=SQL"
   -e "ctp=https;
       adr='port=9999;
         trusted_certificates=c:\asa90\win32\rsaroot.crt;
         certificate_name=RSA Server'"
```

On an UltraLite client, the equivalent implementation is:

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

certificate_unit

Function

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

Separately licensable option required
Transport-layer security requires that you obtain a separately-licensable SQL Anywhere Studio security option and is subject to export regulations.

☞ To order this component, see "Separately-licensable components" [*Introducing SQL Anywhere Studio*, page 5].

Syntax

certificate_unit=*organization-unit*

Protocols

- ♦ dbmlsync - TCP/IP (via the security option), HTTPS, HTTPS_FIPS
- ♦ UltraLite - TCP/IP (via security parms), HTTPS, UlSecureRSASocketStream, UlSecureSocketStream

Default

None

Remarks

For TCP/IP, set this option as part of the security option. For more information, see "security" on page 50.

HTTPS_FIPS is available on Windows only.

☞ For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 106.

☞ For information about how to set network protocol options with UltraLite, see "Network protocol options for UltraLite synchronization clients" on page 341.

See also

- ♦ "MobiLink Transport-Layer Security" [*MobiLink Administration Guide*, page 165]
- ♦ "Verifying certificate fields" [*MobiLink Administration Guide*, page 180]
- ♦ "-x option" [*MobiLink Administration Guide*, page 214]
- ♦ "security" on page 50
- ♦ "trusted_certificates" on page 53
- ♦ "certificate_company" on page 37
- ♦ "certificate_name" on page 38

Example — For examples of https security, see "certificate_name" on page 38 and "trusted_certificates" on page 53.

client_port

Function — Specify a range of client ports for communication.

Syntax — **client_port=***nnnnn*[**-***mmmmm*]

Protocols

- ♦ dbmlsync - TCP/IP, HTTP, HTTPS, HTTPS_FIPS
- ♦ UltraLite - TCP/IP, HTTP, HTTPS, UlSecureRSASocketStream, UlSecureSocketStream

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 synchronization server outside the firewall.

HTTPS_FIPS is available on Windows only.

☞ For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 106.

☞ For information about how to set network protocol options with UltraLite, see "Network protocol options for UltraLite synchronization clients" on page 341.

custom_header

Function — Specify a custom HTTP header.

Syntax — **custom_header=***header*

HTTP headers are of the form *header_name***:** *header_value.*

Protocols

- ♦ dbmlsync - HTTP, HTTPS, HTTPS_FIPS
- ♦ UltraLite - HTTP and HTTPS, but not supported for Java UltraLite

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.

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

HTTPS_FIPS is available on Windows only.

☞ For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 106.

☞ For information about how to set network protocol options with UltraLite, see "Network protocol options for UltraLite synchronization clients" on page 341.

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

host

Function

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

Syntax

host=*hostname-or-ip*

Protocols

- ♦ dbmlsync - TCP/IP, HTTP, HTTPS, HTTPS_FIPS
- ♦ UltraLite - TCP/IP, HTTP, HTTPS, UlSecureRSASocketStream, UlSecureSocketStream

Default

- ♦ Windows CE - the default value is taken from the desktop machine where the CE device cradle is connected, which is stored as the *ipaddr* entry in the registry folder *Comm\Tcpip\Hosts\ppp_peer*.
- ♦ All other devices - the default is **localhost**.

Remarks

On Windows CE, do not use localhost, which refers to the remote device itself. The default value allows a Windows CE device to connect to a MobiLink synchronization server on the desktop machine where the Windows CE device's cradle is connected.

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

HTTPS_FIPS is available on Windows only.

☞ For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 106.

☞ For information about how to set network protocol options with UltraLite, see “Network protocol options for UltraLite synchronization clients” on page 341.

http_password

Function	Authenticate to third-party HTTP servers and gateways using RFC 2617 Basic or Digest authentication.
Syntax	**http_password=***password*
Protocols	♦ dbmlsync - HTTP, HTTPS, HTTPS_FIPS ♦ UltraLite - HTTP, HTTPS
Default	None

Remarks

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

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

You must use http_userid with this option.

HTTPS_FIPS is available on Windows only.

☞ For information about how to set network protocol options with dbmlsync, see “CommunicationAddress (adr) extended option” on page 106.

☞ For information about how to set network protocol options with UltraLite, see “Network protocol options for UltraLite synchronization clients” on page 341.

See also

- ♦ “http_userid” on page 45
- ♦ “http_proxy_password” on page 43
- ♦ “http_proxy_userid” on page 44

Example

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

```
synch_info.stream = ULHTTPSStream();
    synch_info.stream_parms = TEXT("http_userid=user;http_
         password=pwd");
```

http_proxy_password

Function

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

Syntax **http_proxy_password=***password*

Protocols
- ♦ dbmlsync - HTTP, HTTPS, HTTPS_FIPS
- ♦ UltraLite - HTTP, HTTPS

Default None

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

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

You must use http_proxy_userid with this option.

HTTPS_FIPS is available on Windows only.

☞ For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 106.

☞ For information about how to set network protocol options with UltraLite, see "Network protocol options for UltraLite synchronization clients" on page 341.

See also
- ♦ "http_password" on page 43
- ♦ "http_userid" on page 45
- ♦ "http_proxy_userid" on page 44

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

```
synch_info.stream = ULHTTPSStream();
    synch_info.stream_parms = TEXT("http_proxy_userid=user;http_
         proxy_password=pwd");
```

http_proxy_userid

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

Syntax **http_proxy_userid=***userid*

Protocols
- ♦ dbmlsync - HTTP, HTTPS, HTTPS_FIPS
- ♦ UltraLite - HTTP, HTTPS

Default None

Remarks

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

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

You must use http_proxy_password with this option.

HTTPS_FIPS is available on Windows only.

☞ For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 106.

☞ For information about how to set network protocol options with UltraLite, see "Network protocol options for UltraLite synchronization clients" on page 341.

See also

- "http_password" on page 43
- "http_userid" on page 45
- "http_proxy_password" on page 43

Example

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

```
synch_info.stream = ULHTTPSStream();
    synch_info.stream_parms = TEXT("http_proxy_userid=user;http_
         proxy_password=pwd");
```

http_userid

Function

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

Syntax

http_userid=*userid*

Protocols

- dbmlsync - HTTP, HTTPS, HTTPS_FIPS
- UltraLite - HTTP, HTTPS

Default

None

Remarks

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

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

You must use http_password with this option.

HTTPS_FIPS is available on Windows only.

☞ For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 106.

☞ For information about how to set network protocol options with UltraLite, see "Network protocol options for UltraLite synchronization clients" on page 341.

See also

- "http_password" on page 43
- "http_proxy_password" on page 43
- "http_proxy_userid" on page 44

Example

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

```
synch_info.stream = ULHTTPSStream();
    synch_info.stream_parms = TEXT("http_userid=user;http_
         password=pwd");
```

liveness_timeout

Function

Specify the amount of time, in seconds, after the last communication with the client before MobiLink assumes that the connection to the client has been lost and aborts the synchronization.

Syntax

liveness_timeout=*n*

Protocols

- dbmlsync - TCP/IP
- UltraLite - TCP/IP, UlSecureRSASocketStream, UlSecureSocketStream

Default

120 seconds

Remarks

A value of 0 means that there is no timeout.

This option can be useful if download acknowledgement on the client is set to off (the default). This option is not normally used when download acknowledgement is on because if the download takes longer than n seconds to apply, the MobiLink synchronization server aborts the synchronization. Using a larger liveness_timeout value in this case usually defeats the purpose of the timeout.

☞ For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 106.

☞ For information about how to set network protocol options with UltraLite, see "Network protocol options for UltraLite synchronization clients" on page 341.

network_connect_timeout

Function	Specify the number of seconds before the synchronization client should give up trying to connect.
Syntax	network_connect_timeout=*seconds*
Protocols	♦ dbmlsync - TCP/IP, HTTP, HTTPS, HTTPS_FIPS ♦ UltraLite - TCP/IP, HTTP, HTTPS
Default	120 seconds
Remarks	You must specify network_name to use this option.

This feature applies to Pocket PC 2002 only. On Windows, you control this feature by configuring the connection profile for a given network_name.

HTTPS_FIPS is available on Windows only.

☞ For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 106.

☞ For information about how to set network protocol options with UltraLite, see "Network protocol options for UltraLite synchronization clients" on page 341.

network_leave_open

Function	When you specify network_name, you can optionally specify that the network connectivity should be left open after the synchronization finishes (1).
Syntax	**network_leave_open=**{ **0** | **1** }
Protocols	♦ dbmlsync - TCP/IP, HTTP, HTTPS, HTTPS_FIPS ♦ UltraLite - TCP/IP, HTTP, HTTPS
Default	Network connectivity is closed after synchronization (**0**)
Remarks	You must specify network_name to use this option.

When this option is set to 1, network connectivity is left open after the synchronization finishes.

HTTPS_FIPS is available on Windows only.

☞ For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 106.

☞ For information about how to set network protocol options with

UltraLite, see “Network protocol options for UltraLite synchronization clients” on page 341.

See also

“network_name” on page 48

network_name

Function

Specify the network name to start when not already connected to the network.

Syntax

network_name=*name*

Protocols

- ♦ dbmlsync - TCP/IP, HTTP, HTTPS, HTTPS_FIPS
- ♦ UltraLite - TCP/IP, HTTP, HTTPS

Default

None

Remarks

Specify the network name so that you can use MobiLink’s auto-dial feature. This allows you to connect from a Pocket PC 2002 or Windows desktop computer without manually dialing.

Used with scheduling, your remote can synchronize unattended. Used without scheduling, this allows you to run dbmlsync without manually dialing a connection. The name should be the network name that you have specified in the dropdown list in Settings ➤ Connections ➤ Connections (Pocket PC) or Network & Dialup Connections (Windows).

HTTPS_FIPS is available on Windows only.

☞ For information about how to set network protocol options with dbmlsync, see “CommunicationAddress (adr) extended option” on page 106.

☞ For information about how to set network protocol options with UltraLite, see “Network protocol options for UltraLite synchronization clients” on page 341.

See also

- ♦ “Scheduling synchronization” on page 88
- ♦ “network_connect_timeout” on page 47
- ♦ “network_leave_open” on page 47

persistent

Function

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

Syntax

persistent={ **0** | **1** }

Protocols

- ♦ dbmlsync - HTTP, HTTPS, HTTPS_FIPS
- ♦ UltraLite - HTTP, HTTPS

HTTP, HTTPS

Default	♦ Palm - **0** ♦ All other devices - **1**
Remarks	1 means that the client will attempt to use the same TCP/IP connection for all HTTP requests in a synchronization. A setting of 0 is usually more compatible with intermediate agents. Except on Palm devices, you should only set persistent to 1 if you are connecting directly to MobiLink. If you are connecting through an intermediate agent such as a proxy or redirector, a persistent connection may cause problems. HTTPS_FIPS is available on Windows only. ☞ For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 106. ☞ For information about how to set network protocol options with UltraLite, see "Network protocol options for UltraLite synchronization clients" on page 341.

port

Function	Specify the socket port number.
Syntax	**port=***port-number*
Protocols	♦ dbmlsync - TCP/IP, HTTP, HTTPS, HTTPS_FIPS ♦ UltraLite - TCP/IP, HTTP, HTTPS, UlSecureRSASocketStream, UlSecureSocketStream
Default	For TCP/IP, the default is **2439**, which is the IANA-registered port number for the MobiLink synchronization server. For HTTP, the default is **80**. For HTTPS and HTTPS_FIPS, the default is **443**.
Remarks	The port number must be a decimal number that matches the port the MobiLink synchronization 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. HTTPS_FIPS is available on Windows only. ☞ For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 106. ☞ For information about how to set network protocol options with UltraLite, see "Network protocol options for UltraLite synchronization

clients” on page 341.

proxy_host

Function

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

Syntax

proxy_host=*proxy_hostname_or_ip*

Protocols

- dbmlsync - HTTP, HTTPS, HTTPS_FIPS
- UltraLite - HTTP, HTTPS

HTTP, HTTPS

Default

None

Remarks

Use only if going through a proxy.

HTTPS_FIPS is available on Windows only.

☞ For information about how to set network protocol options with dbmlsync, see “CommunicationAddress (adr) extended option” on page 106.

☞ For information about how to set network protocol options with UltraLite, see “Network protocol options for UltraLite synchronization clients” on page 341.

proxy_port

Function

Specify the port number of the proxy server.

Syntax

proxy_port=*proxy_port_number*

Protocols

- dbmlsync - HTTP, HTTPS, HTTPS_FIPS
- UltraLite - HTTP, HTTPS

Default

None

Remarks

Use only if going through a proxy.

HTTPS_FIPS is available on Windows only.

☞ For information about how to set network protocol options with dbmlsync, see “CommunicationAddress (adr) extended option” on page 106.

☞ For information about how to set network protocol options with UltraLite, see “Network protocol options for UltraLite synchronization clients” on page 341.

security

Function

Specify an encryption cipher and encryption options for synchronization.

Separately licensable option required
Transport-layer security requires that you obtain a separately-licensable SQL Anywhere Studio security option and is subject to export regulations.

☞ To order this component, see "Separately-licensable components" [*Introducing SQL Anywhere Studio,* page 5].

Syntax

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

Protocols

dbmlsync - TCP/IP

Default

None

Remarks

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

♦ **ecc_tls** for elliptic-curve encryption. For backwards compatibility, **ecc_tls** can also be specified as **certicom_tls**.

♦ **rsa_tls** for RSA encryption.

♦ **rsa_tls_fips** for RSA encryption that is FIPS-approved. The rsa_tls_fips cipher uses separate FIPS 140-2 certified software from Certicom. Clients using rsa_tls are compatible with servers using rsa_tls_fips, and clients using rsa_tls_fips are compatible with servers using rsa_tls. rsa_tls_fips can only be used with Adaptive Server Anywhere databases on Windows.

☞ For more information, see "Configuring MobiLink clients to use transport-layer security" [*MobiLink Administration Guide,* page 179].

The following security keywords are supported.

♦ **certificate_company=***organization*

♦ **certificate_name=***common_name*

♦ **certificate_unit=***organization_unit*

♦ **trusted_certificates=***filename*

When using dbmlsync with HTTPS, you do not set the security option, but set these four security keywords directly. For details of how to set the security options, see each security keyword.

For UltraLite, see "Security synchronization parameter" on page 328.

☞ For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 106.

☞ For information about how to set network protocol options with UltraLite, see "Network protocol options for UltraLite synchronization clients" on page 341.

See also

- ♦ "MobiLink Transport-Layer Security" [*MobiLink Administration Guide*, page 165]
- ♦ "-x option" [*MobiLink Administration Guide*, page 214]
- ♦ "certificate_company" on page 37
- ♦ "certificate_name" on page 38
- ♦ "certificate_unit" on page 40
- ♦ "trusted_certificates" on page 53

Example

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

```
dbmlsrv9
  -c "dsn=asa90sample;uid=DBA;pwd=SQL"
  -x tcpip(
    port=9999;
    security=rsa_tls(
      certificate=c:\asa90\win32\rsaserver.crt;
      certificate_password=test))
dbmlsync
  -c "dsn=mydb;uid=DBA;pwd=SQL"
  -e "ctp=tcpip;
      adr='port=9999;
         security=rsa_tls(
           trusted_certificates=c:\asa90\win32\rsaroot.crt;
           certificate_name=RSA Server)'"
```

For UltraLite clients, you implement the client side slightly differently. The equivalent client implementation in an Embedded SQL or C++ UltraLite application is:

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

set_cookie

Function

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

Syntax

set_cookie=*cookie_name***=***cookie_value* [**,***cookie_name***=***cookie_value*, ...]

Protocols

- ♦ dbmlsync - HTTP, HTTPS, and HTTPS_FIPS
- ♦ UltraLite - HTTP and HTTPS, but not supported for Java UltraLite

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.

HTTPS_FIPS is available on Windows only.

☞ For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 106.

☞ For information about how to set network protocol options with UltraLite, see "Network protocol options for UltraLite synchronization clients" on page 341.

Example

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

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

trusted_certificates

Function

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

Separately licensable option required
Transport-layer security requires that you obtain the separately-licensable SQL Anywhere Studio security option and is subject to export regulations.

☞ To order this component, see "Separately-licensable components" [*Introducing SQL Anywhere Studio*, page 5].

Syntax

trusted_certificates=*filename*

Protocols

- dbmlsync - TCP/IP (via the security option), HTTPS, HTTP_FIPS
- UltraLite - TCP/IP (via security parms), HTTPS. Not supported on Palm OS.

Default

None

Remarks

When synchronization occurs through a Certicom TLS synchronization stream, the MobiLink synchronization server sends its certificate to the client, as well as 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.

This option is required when using encryption for dbmlsync.

This option is not available on the Palm platform or with UltraLite Static Java.

For TCP/IP, set this option as part of the security option. For more information, see "security" on page 50.

HTTPS_FIPS is available on Windows only.

☞ For information about how to set network protocol options with dbmlsync, see "CommunicationAddress (adr) extended option" on page 106.

☞ For information about how to set network protocol options with UltraLite, see "Network protocol options for UltraLite synchronization clients" on page 341.

See also

- "MobiLink Transport-Layer Security" [*MobiLink Administration Guide*, page 165]
- "-x option" [*MobiLink Administration Guide*, page 214]
- "security" on page 50
- "certificate_company" on page 37
- "certificate_name" on page 38
- "certificate_unit" on page 40

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.

```
dbmlsrv9
   -c "dsn=asa90sample;uid=DBA;pwd=SQL"
   -x https(
     port=9999;
     certificate=c:\asa90\win32\rsaserver.crt;
     certificate_password=test)
```

On an Adaptive Server Anywhere client, the implementation is:

```
dbmlsync
   -c "dsn=mydb;uid=DBA;pwd=SQL"
   -e "ctp=https;
       adr='port=9999;
         trusted_certificates=c:\asa90\win32\rsaroot.crt;
         certificate_name=RSA Server'"
```

On an UltraLite client, the equivalent implementation is:

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

For UltraLite clients addressing the Palm OS, the following command embeds the trusted root certificate in the generated code with the UltraLite Generator:

```
ulgen.exe -c "dsn=asa90sample;uid=DBA;pwd=SQL"  -r c:\asa90\
         win32\rsaroot.crt test test.c
```

Alternatively, you can use the UltraLite initialization utility:

```
ulinit.exe -c "dsn=asa90sample;uid=DBA;pwd=SQL" -t c:\asa90\
         win32\rsaroot.crt -f test.usm -n test_pub
```

url_suffix

Function

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

Syntax

url_suffix=*suffix*

Protocols

- ♦ dbmlsync - HTTP, HTTPS, HTTPS_FIPS
- ♦ UltraLite - HTTP, HTTPS

HTTP, HTTPS

Default

The default value is *MobiLink*.

Remarks

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

☞ For information about how to set this option when using the Redirector, see “Configuring MobiLink clients and servers for the Redirector” [*MobiLink Administration Guide*, page 137].

HTTPS_FIPS is available on Windows only.

☞ For information about how to set network protocol options with dbmlsync, see “CommunicationAddress (adr) extended option” on page 106.

☞ For information about how to set network protocol options with UltraLite, see “Network protocol options for UltraLite synchronization clients” on page 341.

version

Function	Specify the version of HTTP to use for synchronization.
Syntax	**version=***HTTP_version_number*
Protocols	♦ dbmlsync - HTTP, HTTPS, HTTPS_FIPS ♦ UltraLite - 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**. HTTPS_FIPS is available on Windows only. ☞ For information about how to set network protocol options with dbmlsync, see “CommunicationAddress (adr) extended option” on page 106. ☞ For information about how to set network protocol options with UltraLite, see “Network protocol options for UltraLite synchronization clients” on page 341.

PART II

ADAPTIVE SERVER ANYWHERE CLIENTS

This part contains material that describes how to set up and run Adaptive Server Anywhere clients for MobiLink synchronization.

CHAPTER 5

Adaptive Server Anywhere Clients

About this chapter

This chapter describes how to use Adaptive Server Anywhere databases as MobiLink clients.

☞ For a tutorial to walk you through some of the concepts in this chapter, see "Tutorial: Introduction to MobiLink" [*MobiLink Tutorials,* page 1].

☞ For information about UltraLite databases as MobiLink clients, see"UltraLite Clients" on page 277.

Contents

Creating a remote database

Any Adaptive Server Anywhere database can be converted for use as a remote database in a MobiLink installation. All you need to do is create a publication, create a MobiLink user, and subscribe the MobiLink user to the publication.

❖ To create an Adaptive Server Anywhere remote database

1. Start with an existing Adaptive Server Anywhere database, or create a new one and add your tables.
2. Create one or more publications in the new database.

   ☞ See "Publishing data" on page 64.
3. Create a MobiLink user.

   ☞ See "Creating MobiLink users" on page 71.
4. Subscribe a MobiLink user to one or more of the publications.

   ☞ See "Subscribing MobiLink synchronization users" on page 75.

Deploying remote databases

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

❖ To deploy MobiLink remote databases by customizing a prototype

1. Create a prototype remote database.

   The prototype database should have all the tables and publications needed, but not the information that is specific to each database. This individual 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, carry out the following operations:
   - Create a directory to hold the remote database.
   - Copy the prototype remote database into the directory.

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

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

  The SQL script can be a parameterized script. For information on parameterized scripts, see "PARAMETERS statement [Interactive SQL]" [*ASA SQL Reference*, page 561], and "Using SQL command files" [*ASA SQL User's Guide*, page 596].

Example

The following SQL script is taken from the Contact sample. It can be found in *Samples\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'
        OPTION MEM=''
go
CREATE SYNCHRONIZATION SUBSCRIPTION TO "DBA"."Product"
        FOR {ml_userid}
go
CREATE SYNCHRONIZATION SUBSCRIPTION TO "DBA"."Contact"
        FOR {ml_userid}
go
commit work
go
```

The following command line executes the script for a remote database with data source **dsn_remote_1**.

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

Partitioning data between remote databases

It is common for remote databases to fall into separate categories, each with their own requirements. Consider a sales application. All the sales personnel in one region may require access to a particular set of data, but not require access to information about regions other than their own. Employees in other departments may require data of an entirely different nature. Managers may require data that should not be accessible to their subordinates.

Publications are typically used to specify fundamentally different sets of data. For example, you can create one publication for the sales staff and another publication for those employees who do technical support.

You can further fine-tune the data any given remote database will receive by using a WHERE clause within the publication. This feature is useful when remote databases require similar types of information. For example, it can be used to provide sales representatives with only the information relevant to their region.

☞ For more information, see "Partitioning rows among remote databases" [*MobiLink Administration Guide*, page 52].

Upgrading remote databases

If you install a new Adaptive Server Anywhere remote database over an older version, the synchronization progress information in the consolidated database is incorrect.

You can correct this problem by setting the progress column of the ml_user table to 0 (zero) for this user. This is an exceptional case when direct modification of the MobiLink system tables is required. In other cases, you should not directly access the MobiLink system tables.

☞ For more information, see "Upgrading Adaptive Server Anywhere MobiLink clients" [*What's New in SQL Anywhere Studio*, page 237].

Progress offsets

The offset, also called the progress or state, refers to a position in the transaction log of the remote database. It indicates the point to which all operations for the subscription have been uploaded and acknowledged. Dbmlsync uses the offset to decide what data to upload. On the remote database, the offset is stored in the progress column of the SYS.SYSSYNC system table. On the consolidated database, the offset is stored in the progress column of the ml_user table for version 7.x databases, and in the progress column of the ml_subscription table for version 8.0 and up databases.

For each remote, the remote and consolidated databases maintain an offset for every subscription. When a user synchronizes, the offsets are confirmed for all subscriptions that are associated with the user, even if they are not being synchronized at the time. This is required because publications can synchronize 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 with values from the consolidated and then send a new upload based on those offsets. In most cases, this default is appropriate. For example, it is generally appropriate when the consolidated database is restored from backup and the remote transaction log is intact, or when an upload is successful but communication failure prevented an upload acknowledgement from being sent.

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

☞ For more information, see "-r option" on page 146.

First synchronization always works

When you delete and recreate a remote database and then synchronize for the first time, you don't get a progress offset mismatch even though there may be a mismatch with the progress value on the consolidated database. This is because dbmlsync detects that this is a first synchronization and so does not check the offset of the consolidated database, but instead uses the progress offset on the remote database.

Dbmlsync detects a first synchronization when the columns in the remote database system table SYS.SYSSYNC 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, if your first synchronization synchronizes two or more subscriptions in the same upload stream, 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, you specify the dbmlsync -n option with two publications (-n pub1,pub2), and pub1 has synchronized before but pub2 has not. In this case, if you have deleted and recreated a remote database, there may be a progress offset mismatch with the consolidated database.

For more information, see:

- "SYSSYNC system table" [*ASA SQL Reference,* page 740]
- "ml_user" [*MobiLink Administration Guide,* page 531]
- "ml_subscription" [*MobiLink Administration Guide,* page 528]

Publishing data

A publication is a database object that identifies the data that is to be synchronized. A publication consists of articles, which are subsets of a table's columns, rows, or both. Each publication can contain one or more entire tables, or partial tables consisting of selected rows and columns. In a single publication, no table can be included in more than one article.

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

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

Notes about publications

- DBA authority is required to create and drop publications.
- In order to be able to synchronize when you subscribe to multiple publications, the publications must contain the same 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.
- Publications must include all the columns in the primary key of the table that they reference.
- A single publication can publish a subset of columns from a set of tables and use a WHERE clause to select a set of rows to be replicated.
- Views and stored procedures cannot be included in publications.
- Publications and subscriptions are also used by the Sybase message-based replication technology, SQL Remote. SQL Remote requires publications and subscriptions in both the consolidated and remote databases. In contrast, MobiLink publications appear only in Adaptive Server Anywhere remote databases. MobiLink consolidated databases are configured using synchronization scripts.

Publishing whole tables

The simplest publication you can make consists of a single article, which consists of all rows and columns of one or more tables. These tables must already exist.

❖ To publish one or more entire tables (Sybase Central)

1. Connect to the remote database as a user with DBA authority, using the Adaptive Server Anywhere plug-in.
2. Open the Publications folder.
3. From the File menu, choose New ➤ Publication. The Create a New Publication wizard appears.
4. Type a name for the new publication. Click Next.
5. On the Tables tab, select a table from the list of Available Tables. Click Add. The table appears in the list of Selected Tables on the right.
6. Optionally, you may add additional tables. The order of the tables is not important.
7. Click Finish.

❖ To publish one or more entire tables (SQL)

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

Example

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

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

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

```
CREATE PUBLICATION sales (
   TABLE customer,
   TABLE sales_order,
   TABLE sales_order_items,
   TABLE product
)
```

☞ For more information, see the “CREATE PUBLICATION statement” [*ASA SQL Reference*, page 385].

Publishing only some columns in a table

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

Note

If you create two publications that include the same table with different column subsets, then any user who subscribes to both publications will be unable to synchronize.

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

1. Connect to the remote database as a user with DBA authority using the Adaptive Server Anywhere plug-in.
2. Open the Publications folder.
3. From the File menu, choose New ➤ Publication. The Create a New Publication wizard appears.
4. Type a name for the new publication. Click Next.
5. On the Tables tab, select a table from the list of Available Tables. Click Add. The table is added to the list of Selected Tables on the right.
6. On the Columns tab, double-click the table's icon to expand the list of Available Columns. Select each column you want to publish and click Add. The selected columns appear on the right.
7. Click Finish.

❖ To publish only some columns in a table (SQL)

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

Example

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

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

☞ For more information, see the "CREATE PUBLICATION statement" [*ASA SQL Reference*, page 385].

Publishing only some rows in a table

You can create a publication that contains some or all the columns in a table, but only some of the rows. You do so by writing a search condition that matches only the rows you want to publish.

Sybase Central and the SQL language each provide two ways of publishing only some of the rows in a table; however, only one way is compatible with MobiLink.

- **WHERE clause** Compatible with MobiLink. You can use a WHERE clause to include a subset of rows in an article.
- **Subscription expression** Ignored by MobiLink.

In MobiLink, you can use the WHERE clause to exclude the same set of rows from all subscriptions to a publication. All subscribers to the publication upload any changes to the rows that satisfy the search condition.

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

1. Connect to the remote database as a user with DBA authority using the Adaptive Server Anywhere plug-in.
2. Open the Publications folder.
3. From the File menu, choose New ➤ Publication. The Create a New Publication wizard appears.
4. Type a name for the new publication. Click Next.
5. On the Tables tab, select a table from the list of Available Tables. Click Add. The table is added to the list of Selected Tables on the right.
6. On the WHERE Clauses tab, select the table and type the search condition in the lower box. Optionally, you can use the Insert dialog to assist you in formatting the search condition.
7. Click Finish.

❖ To create a publication using a WHERE clause (SQL)

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

Examples

The following statement creates a publication that publishes the id, company_name, city, and state columns of the customer table, for the customers marked as active in the status column.

```
CREATE PUBLICATION pub_customer (
   TABLE customer (
      id,
      company_name,
      city,
      state )
   WHERE status = 'active'
)
```

In this case, the status column itself is not published. All unpublished rows must have a default value. Otherwise, an error occurs when rows are downloaded for insert from the consolidated database.

The following example creates a single-article publication that includes order information for sales rep number 856.

```
CREATE PUBLICATION pub_orders_samuel_singer (
   TABLE sales_order WHERE sales_rep = 856
)
```

☞ For more information, see the "CREATE PUBLICATION statement" [*ASA SQL Reference*, page 385]. Note that the CREATE PUBLICATION statement includes a SUBSCRIBE BY clause. This clause can be used to selectively publish rows in SQL Remote. However, it is ignored during MobiLink synchronization.

Altering existing publications

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

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

Notes

- Publications can be altered only by the DBA or the publication's owner.
- Be careful. In a running MobiLink setup, altering publications may cause errors and can lead to loss of data.

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

1. Connect to the remote database as a user who owns the publication or as a user with DBA authority.

2. In the left pane, click the publication or article. The properties will appear in the right pane.
3. Configure the desired properties.

❖ To add articles (Sybase Central)

1. Connect to the remote database as a user who owns the publication or as a user with DBA authority using the Adaptive Server Anywhere plug-in.
2. Open the Publications folder.
3. Select a publication.
4. From the File menu, choose New ➤ Article. The Create a New Article wizard appears.
5. In the Article Creation wizard, do the following:
   - ♦ On the first page, select a table.
   - ♦ On the next page, select the number of columns.
   - ♦ On the final page, enter a WHERE clause (if desired).
6. Click Finish to create the article.

❖ To remove articles (Sybase Central)

1. Connect to the database as a user who owns the publication or as a user with DBA authority using the Adaptive Server Anywhere plug-in.
2. Open the Publications folder.
3. Click the publication.
4. In the right pane, right-click the article you want to delete and choose Delete from the popup menu.

❖ To modify an existing publication (SQL)

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

Example

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

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

☞ See also the "ALTER PUBLICATION statement" [*ASA SQL Reference*, page 280].

Dropping publications

You can drop a publication using either Sybase Central or the DROP PUBLICATION statement. Before dropping the publication, you must drop all subscriptions connected to it.

You must have DBA authority to drop a publication.

❖ To delete a publication (Sybase Central)

1. Connect to the remote database as a user with DBA authority using the Adaptive Server Anywhere plug-in.
2. Open the Publications folder.
3. Right-click the desired publications and choose Delete from the popup menu.

❖ To delete a publication (SQL)

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

Example

The following statement drops the publication named pub_orders.

```
DROP PUBLICATION pub_orders
```

☞ See also the "DROP PUBLICATION statement" [*ASA SQL Reference*, page 459].

Creating MobiLink users

A MobiLink user name uniquely identifies a remote database. It is used to identify, and optionally authenticate, clients attempting to connect to the MobiLink synchronization server.

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 Adaptive Server Anywhere is affected by this coincidence.

☞ For information about adding MobiLink users to the consolidated database, see "About MobiLink users" on page 10.

Adding MobiLink users to a remote database

This section describes how to add a MobiLink user name to a remote database. For information on supplying MobiLink user properties, including the password, see "Configuring MobiLink user properties" on page 72.

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

1. Connect to the database from the Adaptive Server Anywhere plug-in as a user with DBA authority.

2. Click the MobiLink Users folder.

3. From the File menu, choose New ➤ MobiLink User. The Create a New MobiLink User wizard appears.

4. Enter a name for the MobiLink user. This name is supplied to the MobiLink synchronization server during synchronization.

   The MobiLink user name uniquely identifies a remote database and so must be unique within your synchronization system.

5. Click Finish.

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

1. Connect to the database as a user with DBA authority.

2. Execute a CREATE SYNCHRONIZATION USER statement. The MobiLink user name uniquely identifies a remote database and so must be unique within your synchronization system.

   The following example adds a MobiLink user named SSinger:

   ```
   CREATE SYNCHRONIZATION USER SSinger
   ```

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

☞ For more information, see "CREATE SYNCHRONIZATION USER statement [MobiLink]" [*ASA SQL Reference*, page 404].

Configuring MobiLink user properties

You can specify properties for each MobiLink user in a remote database.

There are two types of property:

- **Options** Options can be specified on the dbmlsync command line. For a complete list of dbmlsync command line options, see "Adaptive Server Anywhere Client Synchronization Parameters" on page 95.

- **Extended options** Extended options can be specified on the command line, stored in the database, or specified with the sp_hook_dbmlsync_set_extended_options event hook.

  ☞ For a list of extended options, see "dbmlsync extended options" on page 105.

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

1. Connect to the database from the Adaptive Server Anywhere plug-in as a user with DBA authority.

2. Open the MobiLink Users folder.

3. Right-click the MobiLink user name and choose Properties from the pop-up menu.

4. Change the properties as needed.

❖ To store MobiLink extended options in the database (SQL)

1. Connect to the database as a user with DBA authority.

2. Execute an ALTER SYNCHRONIZATION USER statement.

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

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

   ☞ For more information, see "ALTER SYNCHRONIZATION USER statement [MobiLink]" [*ASA SQL Reference*, page 292].

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

☞ For more information, see "CREATE SYNCHRONIZATION USER statement [MobiLink]" [*ASA SQL Reference*, page 404].

❖ To configure MobiLink user properties on the command line

1. You can specify dbmlsync options when you start dbmlsync.

   ☞ For more information, see "Adaptive Server Anywhere Client Synchronization Parameters" on page 95.

❖ To specify MobiLink user properties with a client event hook

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

   ☞ For more information, see "sp_hook_dbmlsync_set_extended_options" on page 225.

Priority order for extended options and connection parameters

The CREATE/ALTER SYNCHRONIZATION USER and CREATE/ALTER SYNCHRONIZATION SUBSCRIPTION statements allow you to store extended options and connection parameters in the database and associate them with subscriptions, users or publications. The dbmlsync utility reads this information from the database.

Note: You specify options for a publication by using the CREATE SYNCHRONIZATION SUBSCRIPTION statement and omitting the FOR clause.

If extended options are specified in both the database and the command line, the option strings are combined. If conflicting options are specified, dbmlsync resolves them as follows, where options specified by 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 on the command line that aren't extended options. (For example, `-ds` overrides `-e "ds=off"`.)

3. options specified on the command line with the dbmlsync -eu option.

4. options specified on the command line with the dbmlsync -e option.

5. options saved with the subscription definition (whether using SQL statements or Sybase Central).

6. options saved with the user definition (whether using SQL statements or Sybase Central).

7. options saved with the publication definition (whether using SQL statements or Sybase Central).

If the connection TYPE or ADDRESS is specified in more than one place, the one specified with the highest priority according to the list above overrides any other specification.

Dbmlsync connection parameters

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

☞ For more information, see:

- ♦ "CommunicationType (ctp) extended option" on page 111
- ♦ "CommunicationAddress (adr) extended option" on page 106
- ♦ "-c option" on page 102

Dropping MobiLink users

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

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

1. Connect to the database from the Adaptive Server Anywhere plug-in as a user with DBA authority.

2. Locate the MobiLink user in the MobiLink Users folder.

3. Right click the MobiLink user and choose Delete from the popup menu.

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

1. Connect to the database as a user with DBA authority.

2. Execute a DROP SYNCHRONIZATION USER statement.

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

   ```
   DROP SYNCHRONIZATION USER SSinger
   ```

   ☞ For more information, see "DROP SYNCHRONIZATION USER statement [MobiLink]" [*ASA SQL Reference*, page 467].

Subscribing MobiLink synchronization users

To complete the setup, you must subscribe at least one MobiLink user to one or more pre-existing publications.

☞ For information about creating publications, see "Publishing data" on page 64. For information about creating MobiLink users, see "Creating MobiLink users" on page 71.

Subscriptions versus synchronization subscriptions
Do not confuse subscriptions (CREATE SUBSCRIPTION statement) with synchronization subscriptions (CREATE SYNCHRONIZATION SUBSCRIPTION statement). Subscriptions work only with SQL Remote. They create relationships between publications and *database users* who have been granted remote privileges. Synchronization subscriptions, used with MobiLink, create relationships between publications and *MobiLink users* .

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

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

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

Example

To synchronize the customer and sales_order tables in the Adaptive Server Anywhere sample database, you could use the following statements.

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

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

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

```
CREATE SYNCHRONIZATION USER demo_ml_user
```

3. To complete the process, subscribe the user to the publication.

```
CREATE SYNCHRONIZATION SUBSCRIPTION TO testpub
   FOR demo_ml_user
   TYPE tcpip
   ADDRESS 'host=localhost;port=2439;'
   OPTION sv='version1'
```

Altering MobiLink subscriptions

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

❖ To alter a synchronization subscription (Sybase Central)

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

❖ To alter a synchronization subscription (SQL)

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

☞ For more information, see "ALTER SYNCHRONIZATION USER statement [MobiLink]" [*ASA SQL Reference*, page 292].

Dropping MobiLink subscriptions

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

You must have DBA authority to drop a synchronization subscription.

❖ To delete a synchronization subscription (Sybase Central)

1. Connect to the database as a user with DBA authority.
2. Open the MobiLink Users folder.
3. Select a MobiLink user.
4. Right-click the desired subscription and choose Delete from the popup menu.

❖ To delete a synchronization subscription (SQL)

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

Example

The following statement drops the synchronization subscription of MobiLink user jsmith to a publication named pub_orders.

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

☞ See also the "DROP SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]" [*ASA SQL Reference*, page 466].

Initiating synchronization

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

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

Connection parameters set in the synchronization subscriptions within the remote database are used to locate the appropriate MobiLink synchronization server.

Permissions for dbmlsync

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

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

Multiple MobiLink synchronization users

Each remote database typically contains exactly one MobiLink synchronization user. In this case, you do not need to specify a MobiLink user name on the dbmlsync command line. However, if the remote database contains more than one, you must specify which MobiLink synchronization user to synchronize using the `-u` command line option.

```
dbmlsync -c "dsn=remote;uid=syncuser" -u mluser
```

Similarly, you can specify the user's password using the `-mp` option, or change the password by specifying the new password with the `-mn` option. These are the user ID and password used to the MobiLink synchronization server and may be different from the user ID and password used to connect to the remote database.

Customizing synchronization

MobiLink provides a number of extended options to customize the synchronization process. Extended options can be set for publications, users, and subscriptions. In addition, extended option values can be overridden using options on the dbmlsync command line.

☞ For a complete list of extended options, see “dbmlsync extended options” on page 105.

❖ To override an extended option on the dbmlsync command line

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

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

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

1. Add the option to the CREATE SYNCHRONIZATION SUBSCRIPTION statement or CREATE SYNCHRONIZATION USER statement in the Adaptive Server 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 stream to 3 Mb and the upload increment size to 3 kb.

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

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

Transaction log files

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

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

☞ For more information, see "Backup and Data Recovery" [*ASA Database Administration Guide,* page 373].

Example

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

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

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

Concurrency during synchronization

To ensure the integrity of synchronizations, dbmlsync must ensure that no rows in the download stream are modified between the time the upload stream is built and the time the download is applied.

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

☞ For more information about locks, see "Types of locks" [*ASA SQL User's Guide,* page 136].

The following options let you customize this locking behavior:

- -d option
- LockTables option

-d option

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

☞ For more information, see "-d option" on page 103.

LockTables option

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

☞ For more information, see "LockTables (lt) extended option" on page 122.

If a conflict is detected, the download phase is cancelled 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 stream and therefore not lost.

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

☞ For more information, see "ConflictRetries (cr) extended option" on page 111.

Initiating synchronization from an application

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

There are two ways to do this:

♦ Use the Dbmlsync Integration Component.

  ☞ For more information, see "Dbmlsync Integration Component" on page 237.

♦ If you are developing in any language that can call a DLL, and are programming in C or C#, you can include the *dbtools.h* header file, located in the *h* subdirectory of your SQL Anywhere directory. This file contains a description of the a_sync_db structure and the DBSynchronizeLog function, which you use to add this functionality to your application. This solution works on all supported platforms, including Windows, UNIX, Linux, and Macintosh.

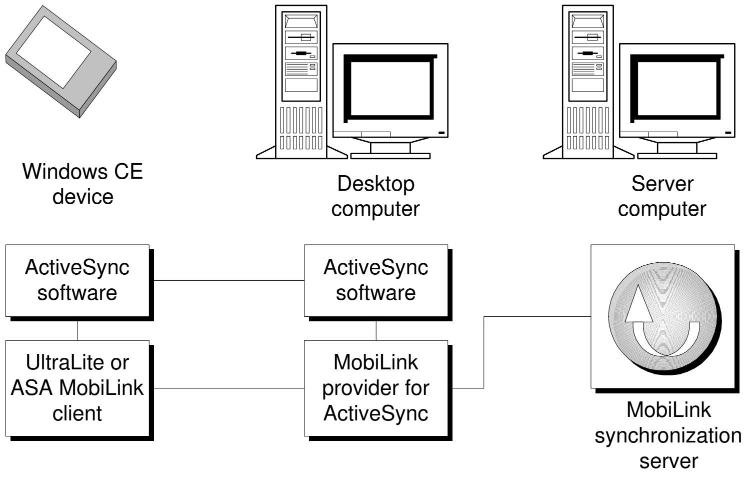
  ☞ For more information, see:

  ♦ "DBTools Interface for dbmlsync" on page 267
  ♦ "DBSynchronizeLog function" [*ASA Programming Guide*, page 275]
  ♦ "a_sync_db structure" [*ASA Programming Guide*, page 298]

Using ActiveSync synchronization

ActiveSync is synchronization software for Microsoft Windows CE handheld devices. Adaptive Server Anywhere MobiLink clients can use ActiveSync version 3.1 or 3.5.

ActiveSync governs synchronization between a Windows CE device and a desktop computer. A MobiLink provider for ActiveSync governs synchronization to the MobiLink synchronization server, as shown in the following diagram.


Windows CE device
Desktop computer
Server computer
ActiveSync software
ActiveSync software
UltraLite or ASA MobiLink client
MobiLink provider for ActiveSync
MobiLink synchronization server

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

- ♦ Configure the Adaptive Server Anywhere remote database for ActiveSync synchronization.

  ☞ See "Configuring Adaptive Server Anywhere remote databases for ActiveSync" on page 83.

- ♦ Install the MobiLink provider for ActiveSync.

  ☞ See "Installing the MobiLink provider for ActiveSync" on page 84.

- ♦ Register the Adaptive Server Anywhere client for use with ActiveSync.

  ☞ See "Registering Adaptive Server Anywhere clients for ActiveSync" on page 85.

If you use ActiveSync synchronization, synchronization must be initiated from the ActiveSync software. The MobiLink provider for ActiveSync can start dbmlsync or it can wake a dbmlsync that is sleeping as scheduled by a schedule string.

You can also put dbmlsync into a sleep mode using a delay hook in the remote database, but the MobiLink provider for ActiveSync cannot invoke synchronization from this state.

☞ For information about scheduling synchronization, see "Scheduling synchronization" on page 88.

Configuring Adaptive Server Anywhere remote databases for ActiveSync

❖ To configure your Adaptive Server Anywhere remote database for ActiveSync

1. Select ActiveSync as the synchronization type.

   The synchronization type can be set for a synchronization publication, for a synchronization user or for a synchronization subscription. It is set in a similar manner for each. Here is part of a typical CREATE SYNCHRONIZATION USER statement:

   ```
   CREATE SYNCHRONIZATION USER SSinger
   TYPE ActiveSync
   ...
   ```

2. Supply an address clause to specify communication between the MobiLink provider for ActiveSync and the MobiLink synchronization server.

   For HTTP or TCP/IP synchronization the ADDRESS clause of the CREATE SYNCHRONIZATION USER or CREATE SYNCHRONIZATION SUBSCRIPTION statement specifies communication between the MobiLink client and server. For ActiveSync, the communication takes place in two stages: from the dbmlsync utility on the device to the MobiLink provider for ActiveSync on the desktop machine, and from desktop machine to the MobiLink synchronization server. The ADDRESS clause specifies the communication between MobiLink provider for ActiveSync and the MobiLink synchronization server.

   The following statement specifies TCP/IP communication to a MobiLink synchronization server on a machine named kangaroo:

   ```
   CREATE SYNCHRONIZATION USER SSinger
   TYPE ActiveSync
   ADDRESS 'stream=tcpip;host=kangaroo;port=2439'
   ```

   ☞ For more information, see "CREATE SYNCHRONIZATION USER statement [MobiLink]" [*ASA SQL Reference*, page 404].

Installing the MobiLink provider for ActiveSync

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

The Adaptive Server Anywhere for Windows CE setup program installs the MobiLink provider for ActiveSync. If you install Adaptive Server Anywhere for Windows CE you do not need to carry out the steps in this section.

When you have installed the MobiLink provider for ActiveSync you must register each application separately. For instructions, see "Registering Adaptive Server Anywhere clients for ActiveSync" on page 85.

❖ To install the MobiLink provider for ActiveSync

1. Ensure that you have the ActiveSync software on your machine, and that the Windows CE device is connected.

2. Enter the following command to install the MobiLink provider:

   ```
   dbasinst -k desk-path -v dev-path
   ```

   where *desk-path* is the location of the desktop component of the provider (*dbasdesk.dll*) and *dev-path* is the location of the device component (*dbasdev.dll*).

   If you have SQL Anywhere installed on your computer, *dbasdesk.dll* is in the *win32* or *win64* subdirectory of your SQL Anywhere directory and *dbasdev.dll* is in a platform-specific directory in the *CE* subdirectory. If you omit -v or -k, these directories are searched by default.

   If you receive a message telling you that the remote provider failed to open, perform a soft reset of the device and repeat the command:

   ☞ For more information, see "ActiveSync provider installation utility" on page 28.

3. Restart your machine.

   ActiveSync does not recognize new providers until the machine is restarted.

4. Enable the MobiLink provider.

   ♦ From the ActiveSync window, click Options.

   ♦ Check the MobiLink item in the list and click OK to activate the provider.

- To see a list of registered applications, click Options again, choose the MobiLink provider, and click Settings.

  ☞ For more information about registering applications, see "Registering Adaptive Server Anywhere clients for ActiveSync" on page 85.

Registering Adaptive Server Anywhere clients for ActiveSync

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

☞ For information on the alternative approach, see "ActiveSync provider installation utility" on page 28.

❖ To register the Adaptive Server Anywhere client for use with ActiveSync

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

   ☞ For information, see "Installing the MobiLink provider for ActiveSync" on page 84.

2. Start the ActiveSync software on your desktop machine.

3. From the ActiveSync window, choose Options.

4. From the list of information types, choose MobiLink and click Settings.

5. In the MobiLink Synchronization dialog, click New. The Properties dialog appears.

6. Enter the following information for your application:

   - **Application name** A name identifying the application to be displayed in the ActiveSync user interface.
   - **Class name** The class name for the dbmlsync client, as set using the -wc option.

     ☞ For more information, see "-wc option" on page 151.
   - **Path** The location of the dbmlsync application on the device.
   - **Arguments** Any command line arguments to be used when ActiveSync starts dbmlsync.

     You start dbmlsync in one of two modes:

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

       ☞ For more information, see "-wc option" on page 151 and "Scheduling synchronization" on page 88.

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

  ☞ For more information, see "-k option" on page 140.

7. Click OK to register the application.

Temporarily stopping synchronization of deletes

Ordinarily, Adaptive Server Anywhere automatically logs any changes to tables or columns that are part of a publication with a synchronization subscription. These changes are uploaded to the consolidated database during the next synchronization.

You may, however, wish to delete rows from synchronized data and not have those changes uploaded. This feature can be used to make unusual corrections, but should be used with caution as it effectively disables part of the automatic synchronization functionality. This technique is a practical alternative to deleting the necessary rows using a download_delete_cursor script

When a STOP SYNCHRONIZATION DELETE statement is executed, none of the delete operations subsequently executed on that connection are synchronized. The effect continues until a START SYNCHRONIZATION DELETE statement is executed. The effects do not nest; that is, subsequent executions of stop synchronization delete after the first will have no additional effect.

❖ To temporarily disable upload of deletes made through a connection

1. Issue the following statement to stop automatic logging of deletes.

   ```
   STOP SYNCHRONIZATION DELETE
   ```

2. Delete rows from the synchronized data, as required, using the DELETE statement. Commit these changes.

3. Restart logging of deletes using the following statement.

   ```
   START SYNCHRONIZATION DELETE
   ```

The deleted rows will not be sent up to the MobiLink synchronization server and hence will not be deleted from the consolidated database.

Scheduling synchronization

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

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

☞ For information about extended options, see "dbmlsync extended options" on page 105 or "-eu option" on page 140. For more information about how to set scheduling, see "Schedule (sch) extended option" on page 127.

❖ To add scheduling to the synchronization subscription

1. 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'
   ```

   ☞ For more information about scheduling syntax, see "Schedule (sch) extended option" on page 127.

   ☞ You can override scheduling instructions and synchronize immediately using the dbmlsync -is option. The -is option instructs dbmlsync to ignore all scheduling information. For more information, see "-is option" on page 140.

❖ To add scheduling from the dbmlsync command line

1. 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.

Hovering

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

☞ For more information, see "HoverRescanThreshold (hrt) extended option" on page 119 and "sp_hook_dbmlsync_log_rescan" on page 215.

Adaptive Server Anywhere version 7 MobiLink clients

Adaptive Server Anywhere 7.0 MobiLink clients were configured using SQL statements that are now deprecated. In particular, synchronization definitions were used instead of publications and subscriptions. The older statements are no longer supported. They have some disadvantages:

1. A synchronization definition is equivalent to a single publication and a single subscription to it. There is no support for subscriptions to multiple publications. In contrast, a single MobiLink user can now subscribe to multiple publications. This allows you to synchronize some portions of your data without synchronizing all of it.

2. Some people found the old terminology confusing. For example, a MobiLink user name was formerly called a site in the context of an Adaptive Server Anywhere client. A MobiLink user is now called a MobiLink user or a synchronization user.

3. The new statements are analogous to those used in SQL Remote, the Sybase message-based replication technology.

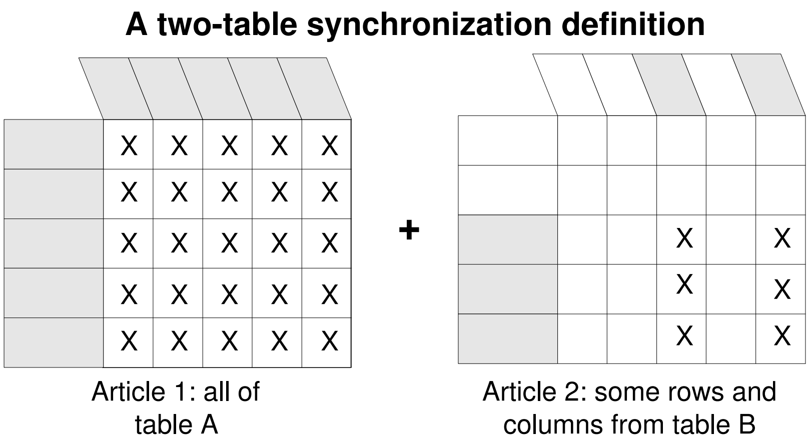
Synchronization definitions identify data to upload in version 7 remote databases

You can choose to synchronize all or any portion of the data in a client Adaptive Server Anywhere database. You can choose to synchronize entire tables, or you can choose to synchronize only particular columns and rows.

The synchronization definition, located in the client Adaptive Server Anywhere database, describes the data that is to be replicated and the location of the appropriate MobiLink synchronization server.

Synchronization scripts, stored in the consolidated database, control how the uploaded rows are processed and which rows are downloaded to the remote database. These scripts do not depend on the type of remote database.

A synchronization definition may include data from several database tables. Each table’s contribution to a synchronization definition is called an *article* . Each article may consist of a whole table, or a subset of the rows and columns in a table.


A two-table synchronization definition
X
+
Article 1: all of table A
Article 2: some rows and columns from table B

Synchronizing a remote database

Once a remote database is set up, the two databases must be periodically brought to a state where they both have the same set of information. This process of synchronization is carried out using the dbmlsync command line utility.

Altering a synchronized table

A table, once added to a synchronization definition, should not be altered. Altering the table interferes with the synchronization process. Should it be necessary to make such an alteration, this step should be performed immediately following synchronization.

The only way to ensure that the ALTER STATEMENT is executed immediately following synchronization is to place this statement in a script, then execute that script using the `-I` option of the *dbmlsync* command line utility.

Comparison to UltraLite clients

If you have developed UltraLite applications for use as MobiLink clients, the following information may be helpful. Many of the elements of a synchronization definition have an UltraLite counterpart.

Adaptive Server Anywhere 9.0 client	Adaptive Server Anywhere 7.0 client	UltraLite clients	MobiLink synchronization server
MobiLink synchronization user	site	user name	MobiLink user
type	type	stream	connection type
address	address	connection parameters	the server's address
script version	script version	version	script version
publication	part of a definition in a remote database, or part of a template in a reference database	*none* —all tables are synchronized	publication
subscription	part of a definition in a remote database, or a part of a site in a reference database	*none*	*none*

Writing synchronization definitions

The synchronization definition is a version 7.0 database object describing data in an Adaptive Server Anywhere remote database that is to be synchronized with a particular MobiLink synchronization server. When using Adaptive Server Anywhere 9.0 or later, publications and synchronization subscriptions should be used instead.

☞ For details, see "Creating a remote database" on page 60.

A synchronization definition should appear only in an Adaptive Server Anywhere 7.0 remote database. MobiLink consolidated servers are configured using scripts.

A synchronization definition specifies the following pieces of information

- ♦ **name** The name of the synchronization definition, known only within the remote database.
- ♦ **site** A name that uniquely identifies this particular MobiLink client.
- ♦ **type** The type of stream to be used to communicate with the MobiLink synchronization server.
- ♦ **address** The parameters necessary to connect to the MobiLink synchronization server.

- **script version** The version of the synchronization scripts the MobiLink synchronization server is to use when synchronizing this client.
- **articles** A description of the data to be synchronized. You can synchronize entire tables, or only particular rows and columns.

The following statement creates a synchronization definition named testpub that defines what data is to be synchronized with site demo_sync_site.

```
CREATE SYNCHRONIZATION DEFINITION testpub
   SITE 'demo_sync_site'
   TYPE 'tcpip'
   ADDRESS 'host=localhost;port=2439;'
   OPTION sv='version1'
   (table People( person_id, fname, lname ),table Pets);
```

In this statement,

- The name of this synchronization definition is testpub. This name is only known within the remote database.
- The name demo_sync_site uniquely identifies this client to the MobiLink synchronization server. This name should appear in the ml_user MobiLink system table, located in the consolidated database.
- The synchronization is to occur over a TCP/IP connection. The connection parameters appear in a string in the ADDRESS clause.

  The TCP/IP connection parameters show that the MobiLink synchronization server is listening on port 2439 of the current machine. Only the listed columns of the People table are synchronized. The option clause is included to indicate that the MobiLink synchronization server should use *version1* of the synchronization scripts when processing data from this client. The default value of this parameter is *default* . Notice that the list of columns is also enclosed in parentheses.
- The MobiLink synchronization server is to use the set of synchronization scripts identified by the name version1 when synchronizing this client. This script version name should appear in the ml_script_version MobiLink system table, located in the consolidated database.
- All columns and rows of the *Pets* table and the listed columns of the *People* table are to be synchronized.

Synchronizing with multiple servers

To synchronize a remote database with multiple MobiLink synchronization servers, create multiple synchronization definitions within the remote database. Each synchronization definition must have a unique site name because, from the point of view of the MobiLink synchronization server, each is a separate logical client.

Synchronizing the same data in one remote database with multiple MobiLink synchronization servers is not presently supported.

Rewriting synchronization definitions for version 8 and up

To use an Adaptive Server Anywhere 7 database as a MobiLink client, you use a synchronization definition to identify which data to upload. In version 8.0 and later, these are better rewritten as publications and synchronization subscriptions.

Example

Suppose you wanted to synchronize the Customer and Sales_Order tables of the sample database. You could have created the following synchronization definition.

```
CREATE SYNCHRONIZATION DEFINITION testpub
   SITE 'demo_ml_user'
   TYPE 'tcpip'
   ADDRESS 'host=localhost;port=2439;'
   OPTION sv='version1'
   (TABLE Customer, TABLE Sales_Order);
```

Instead, you should now do the following.

1. First, publish the Customer and Sales_Order tables.

   ```
   CREATE PUBLICATION testpub
      (TABLE Customer, TABLE Sales_Order);
   ```

2. Next, create a subscription to this publication for the MobiLink user. In this case, the MobiLink user is demo_ml_user. It is unnecessary that a database user of the same name to exist. MobiLink users and database users are independent.

   ```
   CREATE SYNCHRONIZATION SUBSCRIPTION TO testpub
      FOR demo_ml_user
      TYPE 'tcpip'
      ADDRESS 'host=localhost;port=2439;'
      OPTION sv='version1'
   ```

The information is the same, but is broken into two smaller statements instead of one large one.

The SITE clause in the synchronization definition specifies that this particular MobiLink client will synchronizing using the MobiLink user name demo_sync_site. Synchronization is to occur over a TCP/IP connection. The synchronization server is to use the version1 version of the synchronization scripts when interacting with this client.

In the second case, the synchronized tables are published, and then a subscription is created for the demo_sync_site MobiLink user. The TYPE, ADDRESS, and OPTION clauses have the same syntax.

CHAPTER 6

Adaptive Server Anywhere Client Synchronization Parameters

About this chapter

This chapter details all the options you can set for the MobiLink synchronization client, dbmlsync. You can use dbmlsync to synchronize Adaptive Server Anywhere remote databases with a consolidated database.

☞ The dbmlsync utility only works with Adaptive Server Anywhere remote databases. To synchronize UltraLite remote databases, see "UltraLite Synchronization Parameters" on page 315.

Contents

MobiLink synchronization client

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

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

Option	Description
@data	Read in options from the specified environment variable or configuration file. See "@data option" on page 100.
-a	Do not prompt for input again on error. See "-a option" on page 100
-ap	Specify authentication parameters. See "-ap option" on page 100.
-ba *filename*	Apply a download file. See "-ba option" on page 100.
-bc *filename*	Create a download file. See "-bc option" on page 101.
-be *string*	When creating a download file, add a string. See "-be option" on page 101.
-bg	When creating a download file, make it suitable for new remotes. See "-bg option" on page 102.
-c *connection-string*	Supply database connection parameters in the form *parm1=value1*; *parm2=value2*,... If you do not supply this option, a dialog will appear and you must supply connection information. See "-c option" on page 102.
-d	Drop any other connections to the database whose locks conflict with the articles to be synchronized. See "-d option" on page 103.
-dc	Enable restartable downloads. See "-dc option" on page 103.
-dl	Display log messages on the console. See "-dl option" on page 104.
-ds	Specify download-only synchronization. See "-ds option" on page 104.

Option	Description
-e "*option=value*"...	Specify extended options. See "dbmlsync extended options" on page 105.
-eh	Ignore errors that occur in hook functions.
-ek *key*	Specify encryption key. See "-ek option" on page 139.
-ep	Prompt for encryption key. See "-ep option" on page 140.
-eu	Specify extended options for upload defined by most recent `-n` option. See "-eu option" on page 140.
-is	Ignore schedule. See "-is option" on page 140.
-k	Close window on completion. See "-k option" on page 140.
-l	List available extended options. See "-l option" on page 141.
-mn *password*	Specify new MobiLink password. See "-mn option" on page 141.
-mp *password*	Specify MobiLink password. See "-mp option" on page 141.
-n *name*	Specify synchronization publication name(s). See "-n option" on page 142.
-o *logfile*	Log output messages to this file. See "-o option" on page 142.
-os *size*	Specifies a maximum size for the output log, at which point the log is renamed. See "-os option" on page 142.
-ot *logfile*	Truncate file and log output messages to file. See "-ot option" on page 143.
-p	Disable logscan polling. See "-p option" on page 143.
-pd *dllname*;...	Preload specified dlls for Windows CE. See "-pd option" on page 144.
-pi	Test that you can connect to MobiLink. See "-pi option" on page 144.

Option	Description
-pp *number*	Set logscan polling period. See "-pp option" on page 145.
-q	Run in minimized window. See "-q option" on page 146.
-r[**a** | **b**]	Upload retry on client progress. See "-r option" on page 146.
-sc	Reload schema information before each synchronization. See "-sc option" on page 147.
-tu	Remote transactions are preserved on upload. See "-tu option" on page 147.
-u *ml_username*	Allows you to specify the MobiLink user to synchronize. See "-u option" on page 148.
-uo	Synchronization will be upload-only (no download). See "-uo option" on page 149.
-urc *row-estimate*	Allows you to specify an estimate of the rows that will be uploaded. See "-urc option" on page 149.
-v[*levels*]	Verbose operation. See "-v option" on page 150.
-wc *classname*	Specify a window class name. See "-wc option" on page 151.
-x	Rename and restart the transaction log. See "-x option" on page 151.
transaction-logs-directory	Specify the location of the transaction log. See Transaction Log File, below.

Description

Run dbmlsync on the command line to synchronize an Adaptive Server Anywhere remote database with a consolidated database.

To locate and connect to the MobiLink synchronization server, dbmlsync uses the information on the publication, synchronization user, synchronization subscription, or command line.

Transaction log file The *transaction-logs-directory* is the directory that contains the transaction log for the Adaptive Server Anywhere remote database. There is an active transaction log and transaction log archive files, both of which may be required by dbmlsync to determine what to upload. You must specify this parameter if the following are true:

♦ the working log file has been truncated and renamed since you last synchronized

♦ you run the dbmlsync utility from a directory other than the one where the renamed log files are stored

☞ For more information, see "Transaction log files" on page 79.

dbmlsync event hooks There are also dbmlsync client stored procedures that can help you customize the synchronization process. For more information, see "Customizing the client synchronization process" on page 177 and "Dbmlsync Client Event Hooks" on page 175.

Using dbmlsync For more information about using dbmlsync, see "Initiating synchronization" on page 78.

dbmlsync options

This section lists MobiLink synchronization client command line options.

@data option

Function

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

Syntax

dbmlsync *@data* . . .

Description

With this option, you can put command line options in an environment variable or configuration file. If both exist with the name you specify, the environment variable is used.

☞ For more information about configuration files, see "Using configuration files" [*ASA Database Administration Guide,* page 495].

If you want to protect passwords or other information in the configuration file, you can use the File Hiding utility to obfuscate the contents of the configuration file.

☞ See "Hiding the contents of files using the dbfhide command-line utility" [*ASA Database Administration Guide,* page 524].

-a option

Function

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

Syntax

dbmlsync -a . . .

-ap option

Function

Specifies parameters for authentication.

Syntax

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

Description

Use when you use the authenticate_parameters connection script. For example,

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

☞ For more information, see "authenticate_parameters connection event" [*MobiLink Administration Guide,* page 334].

-ba option

Function

Applies a download file.

Syntax **dbmlsync -ba "*filename*"** . . .

Description Specify the name of an existing download file. You can optionally specify a path. If you do not specify a path, the default location is dbmlsync's current working directory (the directory where dbmlsync was started).

☞ For more information, see "File-Based Downloads" [*MobiLink Administration Guide*, page 85].

-bc option

Function Creates a download file.

Syntax **dbmlsync -bc "*filename*"** . . .

Description 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 dbmlsync's current working directory, which is the directory where dbmlsync was started.

Optionally, in the same dbmlsync command line as you create the download file, you can use the -be option to specify a string that can be validated at the remote database, and the -bg option to create a download file for new remote databases.

See also "File-Based Downloads" [*MobiLink Administration Guide*, page 85]

"-be option" on page 101

"-bg option" on page 102

-be option

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

Syntax **dbmlsync -bc "*filename*" -be "*string*"** . . .

Description The string can be used for authentication or other purposes. It is verified at the remote database using the sp_hook_dbmlsync_validate_download_file stored procedure.

See also "sp_hook_dbmlsync_validate_download_file" on page 233

"File-Based Downloads" [*MobiLink Administration Guide*, page 85]

"-bc option" on page 101

-bg option

Function

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

Syntax

dbmlsync -bc "*filename***" -bg** ...

Description

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 will not work with remote databases that have not synchronized because the last download timestamp on a new remote is by default January 1, 1900, which will be 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.

You should not apply download files built with the -bg option to remote databases that have already synchronized. The -bg option causes the generation numbers on the remote database to be updated with the value on the consolidated database at the time the download file was created. For remotes that have already synchronized, this means that the remote database is not forced to upload data before applying a download, and so data could be lost.

See also

"-bc option" on page 101

"File-Based Downloads" [*MobiLink Administration Guide,* page 85]

"MobiLink generation numbers" [*MobiLink Administration Guide,* page 93]

"Synchronizing new remotes" [*MobiLink Administration Guide,* page 88]

-c option

Function

Specifies connection parameters for the remote database.

Syntax

dbmlsync -c "*connection-string***"** ...

Description

The connection string must give dbmlsync permission to connect to the Adaptive Server Anywhere remote database. Commonly, a user ID with REMOTE DBA authority is used.

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 dialog appears. You can specify the remaining command line options in the fields of the connection dialog.

For a complete list of connection parameters for connecting to Adaptive Server Anywhere databases, see "Connection parameters" [*ASA Database Administration Guide*, page 176].

-d option

Function

Drops conflicting locks to the remote database.

Syntax

dbmlsync -d . . .

Description

During synchronization, unless the LockTables extended option is set to OFF, all tables involved in the publications being synchronized are locked to prevent any other processes from making changes. If another connection has a lock on one of these tables, the synchronization fails. Specifying this option forces Adaptive Server Anywhere to drop any other connections to the remote database that hold conflicting locks.

See also

"Concurrency during synchronization" on page 80

-dc option

Function

Specifies restartable downloads.

Syntax

dbmlsync -dc . . .

Description

By default, if MobiLink fails during a download it doesn't apply any of the download data to the remote database. However, it stores the part of the download it did receive in a temporary file on the remote device, so that if you specify -dc the next time you start dbmlsync, it can more quickly complete the download. When you specify -dc, dbmlsync restarts the download and attempts to download the part of the previous download that it did not receive. If it is able to download the remaining data, it applies the complete download to your remote database.

If there is any new data to be uploaded when you use -dc, the restartable download will fail.

You can also specify restartable downloads for Adaptive Server Anywhere remote databases with the ContinueDownload extended option or with the sp_hook_dbmlsync_end hook.

See also

“Resuming failed downloads” [*MobiLink Administration Guide*, page 74]

“ContinueDownload (cd) extended option” on page 112

“sp_hook_dbmlsync_end” on page 212

“DownloadReadSize (drs) extended option” on page 116

-dl option

Function

Displays messages in the log file.

Syntax

dbmlsync -dl . . .

Description

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

-ds option

Function

Specifies download-only synchronization.

Syntax

dbmlsync -ds . . .

Description

When download-only synchronization occurs, dbmlsync does not upload any row operations or data. However, it does upload information about the schema and progress offset.

In addition, dbmlsync ensures that changes on the remote are not overwritten during download-only synchronization. It does this by scanning the log to detect rows with operations waiting to be uploaded. If any of these rows is modified by the download stream, the download stream 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 will take an increasingly long time to complete.

When -ds is used, the ConflictRetries setting is ignored. dbmlsync never retries a download-only synchronization. If a download-only synchronization fails, it will continue to fail until a normal synchronization is performed.

See also

“Upload-only and download-only synchronization” [*MobiLink Administration Guide*, page 24]

"DownloadOnly (ds) extended option" on page 115

"UploadOnly (uo) extended option" on page 133 or "-uo option" on page 149

dbmlsync extended options

Function	Specifies extended options.
Syntax	**dbmlsync -e** *extended-option*=*value*; . . .

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

Parameters

Extended options can be specified by their long form or short form. See each option, below, for details.

Description

Options specified on the command line with the -e option apply to all synchronizations requested on the command line. For example, in the following command line, the extended option sv=test applies to the synchronization of both pub1 and pub2.

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

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

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 using Sybase Central, by using the sp_hook_dbmlsync_set_extended_options event hook, or by using the OPTIONS 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)

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.

- options specified in the sp_hook_dbmlsync_set_extended_options event hook
- options specified on the command line with the -eu option
- options specified on the command line with the -e option
- options specified for the subscription (whether by a SQL statement or in Sybase Central)
- options specified for the user (whether by a SQL statement or in Sybase Central)
- options specified for the publication (whether by a SQL statement or in Sybase Central)

You can review extended options in the log and the SYSSYNC system table.

☞ For information on how extended options can be used to tune synchronization, see "Customizing synchronization" on page 78.

For a detailed explanation of each option, see below.

See also

"-eu option" on page 140

"SYSSYNC system table" [*ASA SQL Reference*, page 740]

"sp_hook_dbmlsync_set_extended_options" on page 225

Example

The following dbmlsync command line illustrates how you can set extended options when you start dbmlsync:

```
dbmlsync -e "adr=host=localhost;dir=c:\db\logs"...
```

The following SQL statement illustrates how you can store extended options in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION TO mypub
   FOR mluser
   ADDRESS 'host=localhost'
   OPTION schedule='weekday@11:30am-12:30pm', dir='c:\db\logs'
```

The following dbmlsync command line opens the usage screen that lists options and their syntax:

```
dbmlsync -l
```

CommunicationAddress (adr) extended option

Function

Specifies network protocol options for connecting to the MobiLink server.

Syntax

dbmlsync -e adr=*protocol-option*; ...

Parameters

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

TCP/IP protocol option	For more information, see...
client_port=*nnnnn*[-*mmmmm*]	"client_port" on page 41
host=*hostname*	"host" on page 42
liveness_timeout=*n*	"liveness_timeout" on page 46
network_connect_-timeout=*seconds*	"network_connect_timeout" on page 47
network_leave_open={**0**|**1**}	"network_leave_open" on page 47
network_name=*name*	"network_name" on page 48
port=*portnumber*	"port" on page 49
security=*cipher***(***keyword***=***value*;. . . **)**	"security" on page 50

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

HTTP protocol option	For more information, see...
buffer_size=*number*	"buffer_size" on page 36
client_port=*nnnnn*[-*mmmmm*]	"client_port" on page 41
custom_header=*header*	"custom_header" on page 41
host=*hostname*	"host" on page 42
network_connect_-timeout=*seconds*	"network_connect_timeout" on page 47
network_leave_open={**0**|**1**}	"network_leave_open" on page 47
network_name=*name*	"network_name" on page 48
persistent={**0**|**1**}	"persistent" on page 48
port=*portnumber*	"port" on page 49
proxy_host=*proxy_hostname*	"proxy_host" on page 50
proxy_port=*proxy_portnumber*	"proxy_port" on page 50
set_cookie=*cookie_name* =*cookie_value*	"set_cookie" on page 52

HTTP protocol option	For more information, see...
url_suffix=*suffix*	"url_suffix" on page 55
version=*versionnumber*	"version" on page 56

- **HTTPS or HTTPS_FIPS protocols** The HTTPS protocol uses Certicom RSA security. The HTTPS_FIPS protocol uses separate FIPS 140-2 certified RSA encryption software from Certicom.

  Separately licensable option required
  Transport-layer security requires that you obtain the separately-licensable SQL Anywhere Studio security option and is subject to export regulations.

  ☞ To order this component, see "Separately-licensable components" [*Introducing SQL Anywhere Studio*, page 5].

  If you specify the HTTPS protocol, you can optionally specify the following protocol options:

HTTPS/HTTPS_FIPS protocol option	For more information, see...
buffer_size=*number*	"buffer_size" on page 36
certificate_company=*company_name*	"certificate_company" on page 37
certificate_name=*name*	"certificate_name" on page 38
certificate_unit=*company_unit*	"certificate_unit" on page 40
client_port=*nnnnn*[-*mmmmm*]	"client_port" on page 41
custom_header=*header*	"custom_header" on page 41
host=*hostname*	"host" on page 42
network_connect_timeout=*seconds*	"network_connect_timeout" on page 47
network_leave_open={**0**|**1**}	"network_leave_open" on page 47

HTTPS/HTTPS_FIPS protocol option	**For more information, see...**
network_name=*name*	"network_name" on page 48
persistent={**0**|**1**}	"persistent" on page 48
port=*portnumber*	"port" on page 49
proxy_host=*proxy_hostname*	"proxy_host" on page 50
proxy_port=*proxy_portnumber*	"proxy_port" on page 50
set_cookie=*cookie_name =cookie_value*	"set_cookie" on page 52
trusted_certificates=*filename*	"trusted_certificates" on page 53
url_suffix=*suffix*	"url_suffix" on page 55
version=*versionnumber*	"version" on page 56

♦ **ActiveSync protocol** During ActiveSync synchronization, ActiveSync is used to exchange data with the MobiLink provider for ActiveSync, which resides on the desktop machine. The ActiveSync protocol options describe the communications between the MobiLink provider for ActiveSync and the MobiLink synchronization server.

The address string for ActiveSync takes the following form:

stream=*desktop-protocol***;**[*desktop-protocol-options*]

where:

- *desktop-protocol* is the network protocol to use between the MobiLink provider for ActiveSync and the MobiLink synchronization server. It can be **http**, **https**, **https_fips** or **tcpip**. The default is **tcpip**.
- *desktop-protocol-options* are TCP/IP, HTTP, HTTPS, or HTTPS_FIPS options, as described in the lists above.

Separately licensable option required
Transport-layer security requires that you obtain the separately-licensable SQL Anywhere Studio security option and is subject to export regulations.

☞ To order this component, see "Separately-licensable components" [*Introducing SQL Anywhere Studio*, page 5].

☞ For more information, see "ActiveSync provider installation utility" on page 28.

Description

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

If you are using the Redirector, see "Configuring MobiLink clients and servers for the Redirector" [*MobiLink Administration Guide*, page 137].

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

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

♦ "CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]" [*ASA SQL Reference*, page 402]
♦ "CREATE SYNCHRONIZATION USER statement [MobiLink]" [*ASA SQL Reference*, page 404]

To specify the TCP/IP, HTTP, HTTPS, or HTTPS_FIPS protocol, use the CommunicationType extended option.

☞ For more information, see "CommunicationType (ctp) extended option" on page 111.

See also

"Configuring MobiLink clients and servers for the Redirector" [*MobiLink Administration Guide*, page 137]

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 or TYPE clause. For example,

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   TYPE 'https'
   ADDRESS host='localhost;port=2439'
```

CommunicationType (ctp) extended option

Function

Specifies the type of network protocol to use for connecting to the MobiLink server.

Syntax

dbmlsync -e ctp=*sync-type*; ...

Description

sync-type can be one of **tcpip**, **http**, **https**, **https_fips** or **ActiveSync**. The default is **tcpip**.

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

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

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

- ♦ "CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]" [*ASA SQL Reference*, page 402]
- ♦ "CREATE SYNCHRONIZATION USER statement [MobiLink]" [*ASA SQL Reference*, page 404]

See also

- ♦ "MobiLink Transport-Layer Security" [*MobiLink Administration Guide*, page 165]
- ♦ "CommunicationAddress (adr) extended option" on page 106

Example

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

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

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

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   TYPE 'tcpip'
   ADDRESS host='localhost'
```

ConflictRetries (cr) extended option

Function

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

Syntax **dbmlsync -e cr=***number*; ...

Description -1 specifies that retries should continue indefinitely. The default is **-1**.

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

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

See also "Handling conflicts" [*MobiLink Administration Guide*, page 64]

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

```
dbmlsync -e "cr=5"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION cr='5';
```

ContinueDownload (cd) extended option

Function Specifies restartable downloads.

Syntax **dbmlsync -e cd=**{ **ON** | **OFF** }; ...

Description By default, if MobiLink fails during a download it doesn't apply any of the download data to the remote database. However, it stores the part of the download it did receive in a temporary file on the remote device, so that if you specify -e cd the next time you start dbmlsync, it can more quickly complete the download. When you specify -e cd, 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 will fail.

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

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

See also

"Resuming failed downloads" [*MobiLink Administration Guide*, page 74]

"sp_hook_dbmlsync_set_extended_options" on page 225

"-dc option" on page 103

"sp_hook_dbmlsync_end" on page 212

Example

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

```
dbmlsync -e "cd=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION cd='on';
```

DisablePolling (p) extended option

Function

Disables automatic logscan polling.

Syntax

dbmlsync -e p= { **ON** | **OFF** }; ...

Description

In order to build an upload stream, dbmlsync must scan the transaction log. Usually it does this just before synchronization. However, when synchronizations are scheduled, dbmlsync by default scans the log in the time between scheduled synchronizations; and when the sp_hook_dbmlsync_delay hook is used, dbmlsync by default scans the log in the pause that occurs just before synchronization. This behavior is more efficient because the log is already at least partially scanned when synchronization begins. This default behavior is called logscan polling.

Logscan polling is on by default but only has an effect when synchronizations are scheduled or when sp_hook_dbmlsync_delay hook is used. When in effect, polling occurs at set intervals: dbmlsync scans to the end of the log, waits for the polling period, and then scans any new transactions in the log. By default, the polling period is 1 minute, but it can be changed with the dbmlsync -pp option or the PollingPeriod extended option.

The default is to not disable logscan polling (**OFF**).

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

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

See also

"PollingPeriod (pp) extended option" on page 126

"-p option" on page 143

"-pp option" on page 145

Example

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

```
dbmlsync -e "p=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION p='on';
```

DownloadBufferSize (dbs) extended option

Function

Specifies the size of the download buffer.

Syntax

dbmlsync -e dbs= *number* [**K** | **M**]; ...

Description

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

If you set this option to 0, dbmlsync does not buffer the download stream. If the setting is greater than 0 but less than 4k, dbmlsync uses a 4k buffer size and issues a warning. The default is **32K** on Windows CE, and **1M** on all other operating systems.

Download buffering increases the benefit of eliminating the download acknowledgement because it allows the worker thread to send the download faster.

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

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

page 105.

Example

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

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

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION dbs='32k';
```

DownloadOnly (ds) extended option

Function

Specifies that synchronization should be download-only.

Syntax

dbmlsync -e ds={ **ON** | **OFF** }; ...

Description

When download-only synchronization occurs, dbmlsync does not upload any row operations or data. However, it does upload information about the schema and progress offset.

In addition, dbmlsync ensures that changes on the remote are not overwritten during download-only synchronization. It does this by scanning the log to detect rows with operations waiting to be uploaded. If any of these rows is modified by the download stream, the download stream 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 will take an increasingly long time to complete.

The default is **OFF** (full synchronization of both upload and download).

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

See also

"-ds option" on page 104

"Upload-only and download-only synchronization" [*MobiLink Administration*

Guide, page 24]

Example

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

```
dbmlsync -e "ds=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION ds='ON';
```

DownloadReadSize (drs) extended option

Function

For restartable downloads, specifies the maximum amount of data that may need to be resent after a communications failure.

Syntax

dbmlsync -e drs= *number* [**K** | **M**]; ...

Description

The DownloadReadSize option is only useful when doing restartable downloads.

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

Dbmlsync reads the download stream 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 will be 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 will be lost. Bytes that are lost in this way will be resent when the download is restarted.

In general, larger DownloadReadSize values result in better performance on successful synchronizations but result in more data being resent when an error occurs.

The typical use of this option is to reduce the default size when communication is unreliable.

The default is **32k**.

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

You can also store extended options in the database. For more information

about dbmlsync extended options, see "dbmlsync extended options" on page 105.

See also

- ♦ "Resuming failed downloads" [*MobiLink Administration Guide*, page 74]
- ♦ "ContinueDownload (cd) extended option" on page 112
- ♦ "sp_hook_dbmlsync_end" on page 212
- ♦ "-dc option" on page 103

Example

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

```
dbmlsync -e "drs=100"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION drs='100';
```

ErrorLogSendLimit (el) extended option

Function

Specifies how much of the remote log file dbmlsync should send to the server when synchronization error occurs.

Syntax

dbmlsync -e el=*number* [**K** | **M**]; ...

Description

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 output log that dbmlsync sends to the MobiLink synchronization server when errors occur during synchronization. Set this option to **0** if you don't want any dbmlsync output log to be sent.

If ErrorLogSendLimit is set to be large enough, dbmlsync sends the entire output log messages from the current session to the MobiLink synchronization server. For example, if the output log messages were appended to an old output log file, dbmlsync only sends the new messages generated in the current session. If the total length of new messages is greater than ErrorLogSendLimit, dbmlsync only logs the last part of the newly generated error and log messages up to the specified size.

Note: The size of the output log is influenced by your verbosity settings. You can adjust these using the dbmlsync -v option, or by using dbmlsync extended options starting with "verbose". For more information, see "-v option" on page 150 and -e verbose options, below.

The default is **32K**.

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

Example

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

```
dbmlsync -e "el=32k"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION el='32k';
```

FireTriggers (ft) extended option

Function

Specifies that triggers should be fired on the remote database when the download is applied.

Syntax

dbmlsync -e ft= { **ON** | **OFF** }; ...

Description

The default is **ON**.

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

Example

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

```
dbmlsync -e "ft=off"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION ft='off';
```

HoverRescanThreshold (hrt) extended option

Function

When you are using scheduling, this limits the amount of discarded memory that is allowed to accumulate before a rescan is performed.

Syntax

dbmlsync -e hrt= *number* [**K** | **M**]; ...

Description

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 scheduling options are specified or when more than one dbmlsync -n option is specified, dbmlsync goes into hovering mode. Hovering is a feature that reduces the amount of time spent scanning the log when dbmlsync is started and asked to perform more than one synchronization before shutting down. Hovering can occur only when all the subscriptions to be synchronized involve the same MobiLink user.

While hovering, dbmlsync keeps track of operations read from the log using a system than maintains information first in memory, and then spills it on to disk. As hovering continues, dbmlsync discards memory that has become fragmented. The amount of memory discarded is proportional to the number of operations processed while hovering and the size of the rows involved (not counting BLOBs). Memory is not discarded if the remote database has only one publication for the user being synchronized.

Discarded memory can be recovered after a complete rescan is performed. There are two ways that you can control when memory is recovered: the HoverRescanThreshold extended option and the sp_hook_dbmlsync_log_rescan stored procedure.

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

See also

"sp_hook_dbmlsync_log_rescan" on page 215

Example

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

```
dbmlsync -e "hrt=2m"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION hrt='2m';
```

IgnoreHookErrors (eh) extended option

Function

Specifies that errors that occur in hook functions should be ignored.

Syntax

dbmlsync -e eh= { **ON** | **OFF** }; ...

Description

The default is **OFF**.

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

This option is equivalent to the dbmlsync -eh option.

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

Example

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

```
dbmlsync -e "eh=off"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION eh='off';
```

IgnoreScheduling (isc) extended option

Function

Specifies that scheduling settings should be ignored.

Syntax

dbmlsync -e isc= { **ON** | **OFF** }; ...

Description

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

This option is equivalent to the dbmlsync -is option.

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

You can also store extended options in the database. For more information

about dbmlsync extended options, see “dbmlsync extended options” on page 105.

See also

“Scheduling synchronization” on page 88

“Schedule (sch) extended option” on page 127

Example

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

```
dbmlsync -e "isc=off"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION isc='off';
```

Increment (inc) extended option

Function

Controls the size of incremental uploads.

Syntax

dbmlsync -e inc= *number* [**K** | **M**]; ...

Description

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

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

The value of this option specifies, very approximately, the size of each upload part. The value of the option controls the size of each upload part as follows. Dbmlsync builds the upload stream 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 -tu.

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

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

page 105.

Example

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

```
dbmlsync -e "inc=32000"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION inc='32k';
```

LockTables (lt) extended option

Function

Specifies that articles (table or parts of tables in the publications being synchronized) should be locked before synchronizing.

Syntax

dbmlsync -e lt= { **ON** | **OFF** | **SHARE** | **EXCLUSIVE** }; ...

Description

SHARE means that dbmlsync locks all synchronization tables in shared mode. EXCLUSIVE means that dbmlsync locks all synchronization tables in exclusive mode. For all platforms except Windows CE, ON is the same as SHARE. For Windows CE devices, ON is the same as EXCLUSIVE. The default is **ON**.

Set to OFF to allow modifications during synchronization.

☞ For more information about shared and exclusive locks, see "How locking works" [*ASA SQL User's Guide*, page 135] and "LOCK TABLE statement" [*ASA SQL Reference*, page 546].

☞ For more information about locking tables in MobiLink applications, see "Concurrency during synchronization" on page 80.

When synchronization tables are locked in exclusive mode (the default for Windows CE devices), no other connections can access the tables, and so dbmlsync stored procedures that require a separate connection will not be able to execute if they require access to any of the synchronization tables. The stored procedures that require a separate connection are

♦ sp_hook_dbmlsync_download_com_error

♦ sp_hook_dbmlsync_download_fatal_sql_error

♦ sp_hook_dbmlsync_download_log_ri_violation

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

Example

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

```
dbmlsync -e "lt=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION lt='on';
```

Memory (mem) extended option

Function

Specifies a cache size.

Syntax

dbmlsync -e mem= *number* [**K** | **M**]; ...

Description

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

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

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

See also

"-urc option" on page 149

"Performance tips" [*MobiLink Administration Guide*, page 106]

Example

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

```
dbmlsync -e "mem=2M"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION mem='2m';
```

MirrorLogDirectory (mld) extended option

Function

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

Syntax

dbmlsync -e mld= *filename* ...

Description

This option makes it possible for dbmlsync to delete old mirror log files when either of the following two circumstances occur:

- the offline mirror log is located in a different directory from the mirror transaction log

  or

- dbmlsync is run on a different machine from the remote database engine

In a normal setup, the active mirror log and renamed mirror transaction logs are located in the same directory, and dbmlsync is run on the same machine as the remote database, so this option is not required and old mirror log files are automatically deleted.

Transaction logs in this directory will only be affected if the DELETE_OLD_LOGS database option is set to ON or DELAY.

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

See also

"DELETE_OLD_LOGS option [replication]" [*ASA Database Administration Guide*, page 652]

Example

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

```
dbmlsync -e "mld=c:\tmp\file"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION mld='c:\tmp\file';
```

MobiLinkPwd (mp) extended option

Function

Specifies the MobiLink password.

Syntax

dbmlsync -e mp=*password*; ...

Description

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

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

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

See also

"NewMobiLinkPwd (mn) extended option" on page 125

"-mn option" on page 141

"-mp option" on page 141

Example

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

```
dbmlsync -e "mp=SQL"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION mp='SQL';
```

NewMobiLinkPwd (mn) extended option

Function

Specifies a new password.

Syntax

dbmlsync -e mn=*new-password*; ...

Description

Specifies a password for the MobiLink user whose subscriptions are being synchronized. Use this option when you want to change an existing password. The default is **NULL**.

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

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

page 105.

See also

"MobiLinkPwd (mp) extended option" on page 124

"-mn option" on page 141

"-mp option" on page 141

Example

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

```
dbmlsync -e "mp=oldpassword; mn=newpassword"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION mn='SQL';
```

OfflineDirectory (dir) extended option

Function

Specifies the path containing offline transaction logs.

Syntax

dbmlsync -e dir=*path*; ...

Description

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

Example

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

```
dbmlsync -e "dir=c:\db\logs"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION dir='c:\db\logs';
```

PollingPeriod (pp) extended option

Function

Specifies the logscan polling period.

Syntax

dbmlsync -e pp=*number* [**S** | **M** | **H** | **D**]; ...

Description

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

Logscan polling occurs only when you are scheduling synchronizations or using the sp_hook_dbmlsync_delay hook.

☞ For an explanation of logscan polling, see “DisablePolling (p) extended option” on page 113.

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

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

You can also store extended options in the database. For more information about dbmlsync extended options, see “dbmlsync extended options” on page 105.

See also

“DisablePolling (p) extended option” on page 113

“-pp option” on page 145

“-p option” on page 143

“sp_hook_dbmlsync_delay” on page 191

Example

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

```
dbmlsync -e "pp=5"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION pp='5';
```

Schedule (sch) extended option

Function

Specifies a schedule for synchronization.

Syntax

dbmlsync -e sch=*schedule*; ...

*schedule***=** { **EVERY**:*hhhh*:*mm* | *singleSchedule* | **INFINITE**,. . . }

hhhh : **00**. . . **9999**

mm : **00**. . . **59**

singleSchedule : *day* *@hh***:***mm*[**AM** | **PM**] [**-***hh***:***mm*[**AM** | **PM**]]

hh : **00**. . . **24**

mm : **00**. . . **59**

day :
EVERYDAY | **WEEKDAY** | **MON** | **TUE** | **WED** | **THU** | **FRI** | **SAT** | **SUN** | *dayOfMonth*

dayOfMonth : **0**. . . **31**

Parameters

EVERY The EVERY keyword causes synchronization to occur immediately, and then repeat indefinitely after the specified time period. If the synchronization process takes longer than the specified period, synchronization starts again immediately.

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 synchronization server by synchronizing at exactly the same time.

The interval end time is always interpreted as following the start time. When the time interval includes midnight, it ends on the next day. If dbmlsync is started midway through the interval, synchronization occurs at a random time before the end time.

EVERYDAY EVERYDAY is all seven days of the week.

WEEKDAY WEEKDAY is Monday through Friday.

Days of the week are Mon, Tue, and so on. You may also use the full forms of the day, such as Monday. You must use the full forms of the day names if the language you are using is not English, is not the language requested by the client in the connection string, and is not the language which appears in the server window.

dayOfMonth To specify the last day of the month regardless of the length of the month, set the *dayOfMonth* to 0.

INFINITE The INFINITE keyword causes dbmlsync to run and to scan the log periodically, but not to synchronize, even if synchronizations are scheduled. A synchronization can happen only when initiated by another program. You can use this option in conjunction with the dbmlsync -wc option to wake up dbmlsync and perform a synchronization.

☞ For more information, see "-wc option" on page 151.

Description

If a previous synchronization is still incomplete at a scheduled time, the scheduled synchronization commences when the previous synchronization completes.

The default is no schedule.

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

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

The IgnoreScheduling extended option and the -is option instruct dbmlsync to ignore scheduling, so that synchronization is immediate. For more information, see "IgnoreScheduling (isc) extended option" on page 120.

☞ For more information about scheduling, see "Scheduling synchronization" on page 88.

Example

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

```
dbmlsync -e "sch=WEEKDAY@8:00am,SUN@9:00pm"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION sch='WEEKDAY@8:00am,SUN@9:00pm';
```

ScriptVersion (sv) extended option

Function

Specifies a script version.

Syntax

dbmlsync -e sv=*version-name*; ...

Description

The script version determines which scripts are run by MobiLink on the consolidated database during synchronization. The default script version name is **default**.

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

Example

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

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

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION sv='SysAd001';
```

SendColumnNames (scn) extended option

Function

Specifies that column names should be sent in the upload.

Syntax

dbmlsync -e scn= { **ON** | **OFF** }; ...

Description

Set this option to ON to tell dbmlsync to send column names from the remote database to the server. This option is required when you generate scripts automatically using the dbmlsrv9 -za or -ze options. This option increases the size of your upload, so you probably won't want to use it if you are not using -za or -ze.

The default is **OFF**.

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

See also

"-za option" [*MobiLink Administration Guide*, page 219]

-ze option

Example

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

```
dbmlsync -e "scn=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION scn='on';
```

SendDownloadACK (sa) extended option

Function

Specifies that a download acknowledgement should be sent from the client to the server.

Syntax

dbmlsync -e sa= { **ON** | **OFF** }; ...

Description

Turning the acknowledgement off (the default) can lead to less contention in the consolidated database and also increased throughput due to shorter download transactions. Download transactions are shorter because they are committed or rolled back as soon as possible, since MobiLink doesn't need to keep these transactions open for as long as it takes the remote client to apply the download. Enable client side download buffering to get the most performance out of eliminating the download acknowledgement. It is recommended that SendDownloadAck be set to OFF.

☞ For more information about improving performance by turning off the download acknowledgement, see "Performance tips" [*MobiLink Administration Guide*, page 106].

Note: When SendDownloadAck is set to ON and you are in verbose mode, an acknowledgement line is written to the client log.

The default is **OFF**.

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

Example

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

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

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION sa='off';
```

SendTriggers (st) extended option

Function

Specifies that trigger actions should be sent on upload.

Syntax

dbmlsync -e st= { **ON** | **OFF** }; ...

Description

Cascaded deletes are also considered trigger actions.

The default is **OFF**.

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

Example

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

```
dbmlsync -e "st=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION st='on';
```

TableOrder (tor) extended option

Function

Specifies the order of tables in the upload stream.

Syntax

dbmlsync -e tor= *tables* ; ...

Description

This option allows you to specify the order of tables that are to be uploaded. Specify *tables* as a comma-separated list. You must specify all tables that are to be uploaded. If you list tables that are not included in the synchronization, they are ignored.

Specify table order to ensure referential integrity. For example, if Table1 refers to Table2, then Table2 must be uploaded before Table1.

In the specified table order, no table may have a foreign key that refers to a table that comes after it in the table order, unless your tables have a cyclical foreign key relationship. By default, dbmlsync selects a table order that satisfies this condition.

There are no cases where this option must be used. It is provided for users who for some reason (usually because it makes their scripts simpler on the consolidated database) would like to ensure that tables are uploaded in a specific order.

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

See also

"Referential integrity and synchronization" [*MobiLink Administration Guide,* page 22]

Example

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

```
dbmlsync -e "tor=admin,parent,child"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION tor='admin,parent,child';
```

UploadOnly (uo) extended option

Function

Specifies that synchronization should only include an upload.

Syntax

dbmlsync -e uo={ **ON** | **OFF** } ; ...

Description

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

The default is **OFF**.

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

See also

"Upload-only and download-only synchronization" [*MobiLink Administration Guide,* page 24]

"DownloadOnly (ds) extended option" on page 115

Example

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

```
dbmlsync -e "uo=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION uo='on';
```

Verbose (v) extended option

Function

Specifies full verbosity.

Syntax

dbmlsync -e v= { **ON** | **OFF** } ; ...

Description

This option specifies a high level of verbosity, which may affect performance and should normally be used in the development phase only.

This option is identical to **dbmlsync -v+**. If you specify both -v and the extended options and there are conflicts, the -v option overrides the extended option. If there is no conflict, the verbosity logging options are additive—all options that you specify are used. When logging verbosity is set by extended option, the logging does not take effect immediately, so startup information is not logged. By the time of the first synchronization, the logging behavior is identical between the -v options and the extended options.

☞ For more information, see "-v option" on page 150.

The default is **OFF**.

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

Example

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

```
dbmlsync -e "v=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION v='on';
```

VerboseHooks (vs) extended option

Function

Specifies that messages related to hook scripts should be logged.

Syntax

dbmlsync -e vs= { **ON** | **OFF** } ; ...

Description

This option is identical to **dbmlsync -vs**. If you specify both -v and the extended options and there are conflicts, the -v option overrides the extended option. If there is no conflict, the verbosity logging options are additive—all options that you specify are used. When logging verbosity is set by extended option, the logging does not take effect immediately, so startup information is not logged. By the time of the first synchronization, the logging behavior is identical between the -v options and the extended options.

☞ For more information, see "-v option" on page 150.

The default is **OFF**.

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

See also

"Dbmlsync Client Event Hooks" on page 175

Example

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

```
dbmlsync -e "vs=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION vs='on';
```

VerboseMin (vm) extended option

Function

Specifies that a small amount of information should be logged.

Syntax

dbmlsync -e vm= { **ON** | **OFF** } ; ...

Description

This option is identical to **dbmlsync -v**. If you specify both -v and the extended options and there are conflicts, the -v option overrides the extended option. If there is no conflict, the verbosity logging options are additive—all options that you specify are used. When logging verbosity is set by extended option, the logging does not take effect immediately, so startup information is not logged. By the time of the first synchronization, the logging behavior is identical between the -v options and the extended options.

☞ For more information, see "-v option" on page 150.

The default is **OFF**.

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

Example

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

```
dbmlsync -e "vm=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION vm='on';
```

VerboseOptions (vo) extended option

Function

Specifies that information should be logged about the command line options (including extended options) that you have specified.

Syntax

dbmlsync -e vo= { **ON** | **OFF** } ; ...

Description

This option is identical to **dbmlsync -vo**. If you specify both -v and the extended options and there are conflicts, the -v option overrides the extended option. If there is no conflict, the verbosity logging options are additive—all options that you specify are used. When logging verbosity is set by extended option, the logging does not take effect immediately, so startup information is not logged. By the time of the first synchronization, the logging behavior is identical between the -v options and the extended options.

☞ For more information, see "-v option" on page 150.

The default is **OFF**.

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

You can also store extended options in the database. For more information

about dbmlsync extended options, see "dbmlsync extended options" on page 105.

Example

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

```
dbmlsync -e "vo=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION vo='on';
```

VerboseRowCounts (vn) extended option

Function

Specifies that the number of rows that are uploaded and downloaded should be logged.

Syntax

dbmlsync -e vn= { **ON** | **OFF** } ; ...

Description

This option is identical to **dbmlsync -vn**. If you specify both -v and the extended options and there are conflicts, the -v option overrides the extended option. If there is no conflict, the verbosity logging options are additive—all options that you specify are used. When logging verbosity is set by extended option, the logging does not take effect immediately, so startup information is not logged. By the time of the first synchronization, the logging behavior is identical between the -v options and the extended options.

☞ For more information, see "-v option" on page 150.

The default is **OFF**.

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

Example

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

```
dbmlsync -e "vn=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION vn='on';
```

VerboseRowValues (vr) extended option

Function	Specifies that the values of rows that are uploaded and downloaded should be logged.
Syntax	**dbmlsync -e vr=** { **ON** | **OFF** } ; ...

Description

This option is identical to **dbmlsync -vr**. If you specify both -v and the extended options and there are conflicts, the -v option overrides the extended option. If there is no conflict, the verbosity logging options are additive—all options that you specify are used. When logging verbosity is set by extended option, the logging does not take effect immediately, so startup information is not logged. By the time of the first synchronization, the logging behavior is identical between the -v options and the extended options.

☞ For more information, see "-v option" on page 150.

The default is **OFF**.

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

Example

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

```
dbmlsync -e "vr=on"
```

The following SQL statement illustrates how you can store this option in the database:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION vr='on';
```

VerboseUpload (vu) extended option

Function	Specifies that information about the upload steam should be logged.
Syntax	**dbmlsync -e vu=** { **ON** | **OFF** } ; ...

Description

This option is identical to **dbmlsync -vu**. If you specify both -v and the extended options and there are conflicts, the -v option overrides the extended option. If there is no conflict, the verbosity logging options are additive—all options that you specify are used. When logging verbosity is set by extended option, the logging does not take effect immediately, so startup information is not logged. By the time of the first synchronization, the logging behavior is identical between the -v options and the extended options.

☞ For more information, see "-v option" on page 150.

The default is **OFF**.

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

You can also store extended options in the database. For more information about dbmlsync extended options, see "dbmlsync extended options" on page 105.

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';
```

-eh option

Function

Ignores errors that occur in hook functions.

Syntax

dbmlsync -eh . . .

-ek option

Function

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

Syntax

dbmlsync -ek *key* . . .

Description

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

-ep option

Function Prompt for the encryption key.

Syntax **dbmlsync -ep** . . .

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

-eu option

Function Specifies extended upload options.

Syntax **dbmlsync -n** *publication-name* **-eu** *keyword=value*;. . .

Description Extended options that are specified on the command line with the -eu option apply only to the synchronization specified by the -n option they follow. For example, on the following command line, the extended option sv=test applies only to the synchronization of pub2.

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

For an explanation of how extended options are processed when they are set in more than one place, see "dbmlsync extended options" on page 105.

For a complete list of extended options, see "dbmlsync extended options" on page 105.

-is option

Function Ignores scheduling instructions so that synchronization is immediate.

Syntax **dbmlsync -is** . . .

Description Ignore extended options that schedule synchronization.

☞ For information about scheduling, see "Scheduling synchronization" on page 88.

-k option

Function Shuts down dbmlsync when synchronization is finished.

Syntax **dbmlsync -k** . . .

Description	When -k is used, dbmlsync exits after synchronization is completed if the synchronization was successful or if an output log file was specified using the -o or -ot options.
See also	"-o option" on page 142
	"-ot option" on page 143

-l option

Function	Lists available extended options.
Syntax	**dbmlsync -l** . . .
Description	When used with the dbmlsync command line it shows you available extended options.

-mn option

Function	Supplies a new password for the user being synchronized.
Syntax	**dbmlsync -mn** *password* . . .
Description	Changes the MobiLink user's password.
	☞ For more information, see "Authenticating MobiLink Users" on page 9.
See also	"MobiLinkPwd (mp) extended option" on page 124
	"NewMobiLinkPwd (mn) extended option" on page 125
	"-mp option" on page 141

-mp option

Function	Supplies the password of the user being synchronized.
Syntax	**dbmlsync -mp** *password* . . .
Description	Supplies the password for MobiLink user authentication.
	☞ For more information, see "Authenticating MobiLink Users" on page 9.
See also	"MobiLinkPwd (mp) extended option" on page 124
	"NewMobiLinkPwd (mn) extended option" on page 125
	"-mn option" on page 141

-n option

Function

Names the synchronization publication.

Syntax

dbmlsync -n *pubname* . . .

Description

Name of synchronization publication. 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 stream.

  In this case, if you have set extended options on the publications, only the options set on the first publication in the list are used. Extended options set on subsequent publications are ignored.

- Specify `-n pub1 -n pub2 -n pub3` to upload pub1 in one upload stream, pub2 in another, and pub3 in a third upload stream.

  When successive synchronizations occur very quickly, such as when you specify `-n pub1 -n pub2`, it is possible that dbmlsync may start processing a synchronization when the server is still processing the previous synchronization. In this case, the second synchronization will fail 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.

  ☞ For more information, see "sp_hook_dbmlsync_delay" on page 191.

-o option

Function

Sends output to a log file.

Syntax

dbmlsync -o *filename* . . .

Description

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

See also

"-os option" on page 142

"-ot option" on page 143

-os option

Function

Specifies a maximum size for the output log, at which point the log is renamed.

Syntax **dbmlsync -os** *size* [**K** | **M** | **G**]. . .

Description The *size* is the maximum file size for logging output messages, specified in units of bytes. Use the suffix k, m or g to specify units of kilobytes, megabytes or gigabytes, respectively. By default, there is no size limit. The minimum size limit is 10 kb.

Before the dbmlsync utility logs output messages to a file, it checks the current file size. If the log message will make the file size exceed the specified size, the dbmlsync utility renames the output file to *yymmddxx*.dbr, where *yymmdd* represents the year, month, and day, and *xx* are sequential characters ranging from AA to ZZ.

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

See also "-o option" on page 142

"-ot option" on page 143

-ot option

Function Truncates the log file and appends output messages to it.

Syntax **dbmlsync -ot** *logfile* . . .

Description The functionality is the same as the -o option except the log file is truncated before any messages are written to it.

See also "-o option" on page 142

"-os option" on page 142

-p option

Function Disables logscan polling.

Syntax **dbmlsync -p** . . .

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

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

The default is to not disable logscan polling (**OFF**).

This option is identical to **dbmlsync -e p**.

See also

"DisablePolling (p) extended option" on page 113

"PollingPeriod (pp) extended option" on page 126

"-pp option" on page 145

-pd option

Function

Preload specified dlls for Windows CE.

Syntax

dbmlsync -pd *dllname*;...

Description

When running dbmlsync on Windows CE, you should use the -pd option to specify dlls that need to be loaded. Otherwise, the correct dlls may not be loaded and an error may be generated.

Following are the dlls that need to be loaded for each communication protocol:

Protocol	DLL
TCP/IP	dbmlsock9.dll
HTTP	dbmlhttp9.dll
HTTPS	dbmlhttps9.dll

You should specify multiple dlls as a semicolon-separated list. For example,

```
-pd dbmlsock9.dll;dbmlhttp9.dll
```

-pi option

Function

Pings a MobiLink synchronization server.

Syntax

dbmlsync -pi -c *connection_string* **-e sv=***script_version*
  [**-n** *pubname*] [**-u** *ml_username*]

Description

The ping option allows you to test that your connection information is correct. When you use -pi, dbmlsync does not initiate synchronization.

In order to be able to ping, dbmlsync must have a unique address for the MobiLink synchronization server. This means that you must include connection parameters and a script version, as well as the publication name, MobiLink user name, or both. The publication or user hold connection information for the remote. You need to specify both when the user is subscribed to multiple publications or the publication has multiple users. For example, if there is only one subscription to the publication, you can specify the publication without the user.

When the MobiLink synchronization server receives a ping request, it connects to the consolidated database, authenticates the user, and then sends the authenticating user status and value back to the client (dbmlsync or UltraLite).

If the ping succeeds, the MobiLink server issues an information message. If the ping does not succeed, it issues an error message.

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 ml_user.

The MobiLink synchronization server may execute the following scripts, if they exist:

♦ begin_connection

♦ authenticate_user

♦ authenticate_user_hashed

♦ end_connection

The client cannot synchronize while it is pinging the server.

-pp option

Function

Specifies the frequency of log scans.

Syntax

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

Description

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

Logscan polling occurs only when you are scheduling synchronizations or using the sp_hook_dbmlsync_delay hook.

☞ For an explanation of logscan polling, see “-p option” on page 143.

See also

“PollingPeriod (pp) extended option” on page 126

“DisablePolling (p) extended option” on page 113

“-p option” on page 143

-q option

Function

Starts the MobiLink synchronization client in a minimized window.

Syntax

dbmlsync -q . . .

Description

For Windows operating systems only.

-r option

Function

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

The -rb option can be used when the remote offset 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, used when the remote offset is greater than the consolidated offset, is provided only for very rare circumstances and may cause data loss.

Syntax

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

Description

☞ For information about progress offsets, see “Progress offsets” on page 62.

-rb If the remote database is restored from backup, the default behavior may cause data to be lost. In this case, the first time you run dbmlsync after the remote database is restored, you should specify -rb. When you use -rb, the upload continues from the offset recorded in the remote database if the offset recorded in the remote is less than that obtained from the consolidated database. If you use -rb and the offset in the remote is not less than the offset from the consolidated database, an error is reported and the synchronization is aborted.

The -rb option may result in some data being uploaded that has already been uploaded. This can result in conflicts in the consolidated database and should be handled with appropriate conflict resolution scripts.

-ra The -ra option should be used only in very rare cases. If you use -ra, the upload is retried starting from the offset obtained from the remote database if the remote offset is greater than the offset obtained from the consolidated database. If you use -ra and the offset in the remote is not greater than the offset from the consolidated database, an error is reported and the synchronization is aborted.

The -ra option should be used with care. If the offset mismatch is the result of a restore of the consolidated database, changes that happened in the remote database in the gap between the two offsets are lost. The -ra option may be useful when the consolidated database has been restored from backup and the remote database transaction log has been truncated at the same point as the remote offset. In this case, all data that was uploaded from the remote database is lost from the point of the consolidated offset to the point of the remote offset.

-sc option

Function

Specifies that dbmlsync should reload schema information before each synchronization.

Syntax

dbmlsync -sc. . .

Description

Prior to version 9.0, dbmlsync reloaded schema information from the database before each synchronization. The information that was reloaded includes foreign key relationships, publication definitions, extended options stored in the database, and information about database settings. On slower handheld devices, loading this information typically took 20 seconds. In most cases this 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.

-tu option

Function

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

Syntax

dbmlsync -tu . . .

Description

When you use -tu, you create a **transaction-level upload**: dbmlsync uploads each transaction on the remote database as a distinct transaction. If you change the same row three times on the remote database and commit the change each time, each of the three transactions is applied in the next upload stream. The transactions are uploaded in a single synchronization, on a single connection, but each transaction is committed as soon as it is successfully uploaded.

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

- ♦ MobiLink always applies updates based on foreign key relationships. For example, when data is changed in child and parent tables, MobiLink inserts data into the parent table before the child table, but deletes data from the child before the parent. If your remote operations do not follow this order, the order of operations will be different on the consolidated database.

- Operations within a transaction are coalesced. This means that if you change the same row three times in one transaction, only the final form of the row is uploaded.

Usage

When you do not use -tu, MobiLink coalesces all changes on the remote database into one transaction in the upload stream. This means that if you change the same row three times between synchronizations, regardless of the number of remote transactions, only the final form of the row is uploaded. This default behavior is efficient and is optimal in many situations.

However, in certain situations you may wish to preserve remote transactions on the consolidated database. For example, you may wish to define triggers on the consolidated database that act on transactions as they occur in the remote database. Or you may want the consolidated database to reflect all changes on the remote database, regardless of when synchronization occurs.

In addition, there are advantages to breaking up the upload stream 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 each transaction is applied as it is successfully uploaded, so 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.

When you use -tu, performance will be improved if you start the MobiLink synchronization server with the -us option, which prevents MobiLink from invoking scripts for tables which have no data to upload.

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 every change, because -tu causes MobiLink to replicate all transactions.

You cannot use -tu with the Increment extended option.

-u option

Function

Specifies the MobiLink user name.

Syntax

dbmlsync -u *ml_username* . . .

Description

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

This option should be used in conjunction with -n *publication* to identify the subscription on which dbmlsync should operate. Each subscription is uniquely identified by an *ml_username*, *publication* pair.

You can only specify one user name on the command line. All subscriptions to be synchronized in a single run must involve the same user. The -u option can be omitted if each publication that is specified on the command line with the -n option has only one subscription.

-uo option

Function

Specifies that synchronization will only include an upload, and no download will occur.

Syntax

dbmlsync -uo. . .

Description

During an 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 stream back down, MobiLink sends only an acknowledgement indicating if the upload was successfully committed.

"Upload-only and download-only synchronization" [*MobiLink Administration Guide*, page 24]

"DownloadOnly (ds) extended option" on page 115

"UploadOnly (uo) extended option" on page 133

-urc option

Function

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

Syntax

dbmlsync -urc *row-estimate* . . .

Description

To improve performance, you can specify an estimate of the number of rows that will be uploaded in a synchronization. In general, a higher estimate results in faster uploads but more memory usage.

Synchronization will proceed correctly regardless of the estimate that is specified.

See also

"Memory (mem) extended option" on page 123

"Performance tips" [*MobiLink Administration Guide*, page 106]

-v option

Function

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*] . . .

Description

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

If you specify –v alone, a small amount of information is logged.

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

- ♦ **+** Turn on all logging options except for c and p.
- ♦ **c** Expose the connect string in the log.
- ♦ **p** Expose the password in the log.
- ♦ **n** Log the number of rows that were uploaded and downloaded.
- ♦ **o** Log information about the command line options and extended options that you have specified.
- ♦ **r** Log the values of rows that were uploaded and downloaded.
- ♦ **s** Log messages related to hook scripts.
- ♦ **u** Log information about the upload stream.

There are extended options that have similar functionality to the -v options. If you specify both -v and the extended options and there are conflicts, the -v option overrides the extended option. If there is no conflict, the verbosity logging options are additive—all options that you specify are used. When logging verbosity is set by extended option, the logging does not take effect immediately, so startup information is not logged. By the time of the first synchronization, the logging behavior is identical between the -v options and the extended options.

See also

"Verbose (v) extended option" on page 134

"VerboseHooks (vs) extended option" on page 135

"VerboseMin (vm) extended option" on page 135

"VerboseOptions (vo) extended option" on page 136

"VerboseRowCounts (vn) extended option" on page 137

"VerboseRowValues (vr) extended option" on page 138

-wc option

Function

Specifies a window class name.

Syntax

dbmlsync -wc *class-name* . . .

Description

This option specifies a class name that can be used to poke dbmlsync and wake it up whenever it is in hover mode, such as when scheduling is enabled or when you are using server-initiated synchronization.

In addition, the window class name identifies the application for ActiveSync synchronization. The class name must be given when registering the application for use with ActiveSync synchronization.

See also

- "Registering Adaptive Server Anywhere clients for ActiveSync" on page 85
- "Using ActiveSync synchronization" on page 82
- INFINITE keyword in "Schedule (sch) extended option" on page 127
- "Scheduling synchronization" on page 88

Example

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

-x option

Function

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

Syntax

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

Description

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

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

If backups are not routinely performed at the remote database, the transaction log continues to grow. As an alternative to using the -x option to control transaction log size, you can use an Adaptive Server Anywhere event handler to control the size of the transaction log. For example, the following event handler renames the transaction log at the remote database when its size exceeds 5 Mb. You can use such an event handler together with the DELETE_OLD_LOGS database option to control the space taken up by transaction logs.

```
CREATE EVENT RenameLogLimit
TYPE GrowLog
WHERE event_condition( ’LogSize’ ) > 5
AT REMOTE
HANDLER
BEGIN
 If event_parameter( ’NumActive’ ) <= 1 then
  BACKUP DATABASE DIRECTORY backupdir
  TRANSACTION LOG ONLY
  TRANSACTION LOG RENAME
 end if;
END
```

In this example, NumActive ensures that only one of the stacked handlers is executed.

See also

- “Automating Tasks Using Schedules and Events” [*ASA Database Administration Guide*, page 301]
- “DELETE_OLD_LOGS option [replication]” [*ASA Database Administration Guide*, page 652]
- “CREATE EVENT statement” [*ASA SQL Reference*, page 351]

CHAPTER 7

MobiLink SQL Statements

About this chapter

This chapter presents detailed descriptions of SQL statements in alphabetical order.

Contents

ALTER PUBLICATION statement

Description

Use this statement to alter a publication. In MobiLink, a publication identifies synchronized data in a Adaptive Server Anywhere remote database. In SQL Remote, publications identify replicated data in both consolidated and remote databases.

Syntax

ALTER PUBLICATION [*owner*.]*publication-name alterpub-clause*, . . .

alterpub-clause:
 ADD TABLE *article-description*
| **MODIFY TABLE** *article-description*
| { **DELETE** | **DROP** } **TABLE** [*owner*.]*table-name*
| **RENAME** *publication-name*

owner, *publication-name*, *table-name* : *identifier*

article-description :
table-name [**(** *column-name*, . . . **)**]
[**WHERE** *search-condition*]
[**SUBSCRIBE BY** *expression*]

Usage

This statement is applicable only to MobiLink and SQL Remote.

The ALTER PUBLICATION statement alters a publication in the database. The contribution to a publication from one table is called an **article**. Changes can be made to a publication by adding, modifying, or deleting articles, or by renaming the publication. If an article is modified, the entire specification of the modified article must be entered.

You set options for a MobiLink publication with the ADD OPTION clause in the ALTER SYNCHRONIZATION SUBSCRIPTION statement or CREATE SYNCHRONIZATION SUBSCRIPTION statement.

Permissions

Must have DBA authority, or be the owner of the publication. Requires exclusive access to all tables referred to in the statement.

Side effects

Automatic commit.

See also

- "CREATE PUBLICATION statement" on page 160
- "DROP PUBLICATION statement" on page 167
- "ALTER SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]" on page 156
- "CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]" on page 162
- "sp_add_article procedure" [*SQL Remote User's Guide*, page 381]
- "sp_add_article_col procedure" [*SQL Remote User's Guide*, page 383]

Standards and compatibility

- **SQL/92** Vendor extension.

- **SQL/99** Vendor extension.

Example

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

```
ALTER PUBLICATION pub_contact
   ADD TABLE customer
```

ALTER SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]

Description

Use this statement in an Adaptive Server Anywhere remote database to alter the properties of a subscription of a MobiLink user to a publication.

Syntax

ALTER SYNCHRONIZATION SUBSCRIPTION
TO *publication-name*
[**FOR** *ml_username*, . . .]
[**TYPE** *protocol*]
[**ADDRESS** *protocol-options*]
[**ADD OPTION** *option*=*value*, . . .]
[**MODIFY OPTION** *option*=*value*, . . .]
[**DELETE** { **ALL OPTION** | **OPTION** *option*, . . . }]

ml_username: *identifier*

protocol-type: **http** | **https** | **https_fips** |**tcpip** | **ActiveSync**

protocol-options: *string*

value: *string* | *integer*

Parameters

TO clause Specify the name of a publication.

FOR clause Specify one or more MobiLink user names.

Omit the FOR clause to set the protocol type, protocol options, and extended options for a publication.

☞ For information about how dbmlsync processes options that are specified in different locations, see "Priority order for extended options and connection parameters" on page 73.

TYPE clause This clause specifies the network protocol to use for synchronization. The default protocol is **tcpip**.

☞ For more information about communication protocols, see "CommunicationType (ctp) extended option" on page 111.

ADDRESS clause This clause specifies network protocol options, including the location of the MobiLink synchronization server.

☞ For a complete list of protocol options, see "CommunicationAddress (adr) extended option" on page 106.

ADD OPTION, MODIFY OPTION, DELETE OPTION, AND DELETE ALL OPTION clauses These clauses allow you to add, modify, delete, or delete all extended options. You may specify only one option in each clause.

The values for each option cannot contain the characters "=" or "," or ";".

☞ For a complete list of options, see "dbmlsync extended options" on page 105.

Usage

Use this statement to alter a synchronization subscription within a MobiLink remote or reference database.

Permissions

Must have DBA authority. Requires exclusive access to all tables referred to in the publication.

Side effects

Automatic commit.

See also

- "CREATE PUBLICATION statement" on page 160
- "CREATE SYNCHRONIZATION USER statement [MobiLink]" on page 164

Standards and compatibility

- **SQL/92** Vendor extension.
- **SQL/99** Vendor extension.

Examples

Create a default subscription, which contains default subscription values, for the sales publication (by omitting the FOR clause). Indicate the address of the MobiLink synchronization server and specify that only the Certicom root certificate is to be trusted.

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   ADDRESS 'host=test.internal;port=2439;
      security=ecc_tls'
   OPTION memory='2m';
```

Subscribe MobiLink user ml_user1 to the sales publication. Set the memory option to 3 Mb, rather than the value specified in the default publication.

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR 'ml_user1'
   OPTION memory='3m';
```

ALTER SYNCHRONIZATION USER statement [MobiLink]

Description

Use this statement in an Adaptive Server Anywhere remote database to alter the properties of a MobiLink user.

Syntax

ALTER SYNCHRONIZATION USER *ml_username*
[**TYPE** *protocol-type*]
[**ADDRESS** *protocol-options*]
[**ADD OPTION** *option***=***value*, . . .]
[**MODIFY OPTION** *option***=***value*, . . .]
[**DELETE** { **ALL OPTION** | **OPTION** *option* }]

ml_username: *identifier*

protocol-type: **http** | **https** | **https_fips** | **tcpip** | **ActiveSync**

protocol-options: *string*

value: *string* | *integer*

Parameters

TYPE clause This clause specifies the network protocol to use for synchronization.

☞ For more information about communication protocols, see "CommunicationType (ctp) extended option" on page 111.

ADDRESS clause This clause specifies network protocol options, including the location of the MobiLink synchronization server.

☞ For a complete list of protocol options, see "CommunicationAddress (adr) extended option" on page 106.

ADD OPTION, MODIFY OPTION, DELETE OPTION, AND DELETE ALL OPTION clauses These clauses allow you to add, modify, delete, or delete all extended options. You may specify only one option in each clause.

☞ For a complete list of options, see "dbmlsync extended options" on page 105.

Usage

Use this statement to alter the properties of a synchronization user within a MobiLink remote database.

Permissions

Must have DBA authority. Requires exclusive access to all tables referred to in the publication.

Side effects

Automatic commit.

See also

- "ALTER SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]" on page 156
- "CREATE SYNCHRONIZATION USER statement [MobiLink]" on page 164
- "CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]" on page 162

Standards and compatibility

- **SQL/92** Vendor extension.
- **SQL/99** Vendor extension.

CREATE PUBLICATION statement

Description

Use this statement to create a publication. In MobiLink, a publication identifies synchronized data in UltraLite or Adaptive Server Anywhere remote databases.

Syntax

CREATE PUBLICATION [*owner*.]*publication-name*
(TABLE *article-description*, . . . **)**

owner, *publication-name* : *identifier*

article-description :
  table-name [**(** *column-name*, . . . **)**]
[**WHERE** *search-condition*]
[**SUBSCRIBE BY** *expression*]

Parameters

article-description Publications are built from articles. Each article is a table or part of a table. An article may be a vertical partition of a table (a subset of the table's columns), a horizontal partition (a subset of the table's rows) or a vertical and horizontal partition.

WHERE clause The WHERE clause is a way of defining the subset of rows of a table to be included in an article. It is useful if the same subset is to be received by all subscribers to the publication.

SUBSCRIBE BY clause You can combine WHERE and SUBSCRIBE BY clauses in an article definition, but the SUBSCRIBE BY clause is used only by SQL Remote.

Usage

This statement is applicable only to MobiLink and SQL Remote.

The CREATE PUBLICATION statement creates a publication in the database. A publication can be created for another user by specifying an owner name.

In MobiLink, publications are required in Adaptive Server Anywhere remote databases, and are optional in UltraLite databases. These publications and the subscriptions to them determine which data will be uploaded to the MobiLink synchronization server. You can construct a remote database by creating publications and subscriptions directly. Alternatively, you can create publications and subscriptions in an Adaptive Server Anywhere reference database, which acts as a template for the remote databases, and then construct the remote databases using the MobiLink extraction utility.

You set options for a MobiLink publication with the ADD OPTION clause in the ALTER SYNCHRONIZATION SUBSCRIPTION statement or CREATE SYNCHRONIZATION SUBSCRIPTION statement.

Permissions

Must have DBA authority. Requires exclusive access to all tables referred to in the statement.

Side effects

Automatic commit.

See also

- "ALTER PUBLICATION statement" on page 154
- "DROP PUBLICATION statement" on page 167
- "ALTER SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]" on page 156
- "CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]" on page 162
- "sp_create_publication procedure" [*SQL Remote User's Guide*, page 386]
- Adaptive Server Anywhere MobiLink clients: "Publishing data" on page 64
- UltraLite MobiLink clients: " Choosing data to synchronize" on page 280

Standards and compatibility

- **SQL/92** Vendor extension.
- **SQL/99** Vendor extension.

Example

The following statement publishes all columns and rows of two tables.

```
CREATE PUBLICATION pub_contact (
   TABLE contact,
   TABLE company
)
```

The following statement publishes only some columns of one table.

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

The following statement publishes only the active customer rows by including a WHERE clause that tests the status column of the customer table.

```
CREATE PUBLICATION pub_customer (
   TABLE customer ( id, company_name, city, state )
   WHERE status = 'active'
)
```

CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]

Description

Use this statement in an Adaptive Server Anywhere remote database to subscribe a MobiLink user to a publication.

Syntax

CREATE SYNCHRONIZATION SUBSCRIPTION
TO *publication-name*
[**FOR** *ml_username*, . . .]
[**TYPE** *protocol-type*]
[**ADDRESS** *protocol-options*]
[**OPTION** *option*=*value*, . . .]

ml_username: *identifier*

protocol-type: **http** | **https** | **https_fips** | **tcpip** | **ActiveSync**

protocol-options: *string*

value: *string* | *integer*

Parameters

TO clause Specify the name of a publication.

FOR clause Specify one or more MobiLink user names.

ml_username is a name identifying a remote database. This name must be unique.

☞ For more information about synchronization user names, see "About MobiLink users" on page 10.

Omit the FOR clause to set the protcol type, protocol options, and extended options for a publication.

☞ For information about how dbmlsync processes options that are specified in different locations, see "Priority order for extended options and connection parameters" on page 73.

TYPE clause This clause specifies the network protocol to use for synchronization. The default protocol is **tcpip**.

☞ For more information about network protocols, see "CommunicationType (ctp) extended option" on page 111.

ADDRESS clause This clause specifies network protocol options such as the location of the MobiLink synchronization server. Multiple options must be separated with semi-colons.

☞ For a complete list of protocol options, see "CommunicationAddress (adr) extended option" on page 106.

OPTION clause This clause allows you to set extended options for the subscription. If no FOR clause is provided, the extended options act as default settings for the publication, and are overridden by any extended options set for a synchronization user.

☞ For information about how dbmlsync processes options that are specified in different locations, see "Priority order for extended options and connection parameters" on page 73.

☞ For a complete list of options, see "dbmlsync extended options" on page 105.

Usage

Use this statement to create a synchronization subscription within a MobiLink remote or reference database.

Permissions

Must have DBA authority. Requires exclusive access to all tables referred to in the publication.

Side effects

Automatic commit.

See also

- "CREATE PUBLICATION statement" on page 160
- "CREATE SYNCHRONIZATION USER statement [MobiLink]" on page 164

Standards and compatibility

- **SQL/92** Vendor extension.
- **SQL/99** Vendor extension.

Examples

Create a default subscription, which contains default subscription values, for the sales publication (by omitting the FOR clause). Indicate the address of the MobiLink synchronization server and specify that only the Certicom root certificate is to be trusted.

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   ADDRESS 'host=test.internal;port=2439;
      security=ecc_tls'
   OPTION memory='2m';
```

Subscribe MobiLink user ml_user1 to the sales publication. Set the memory option to 3 Mb, rather than the value specified in the default publication.

```
CREATE SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR ml_user1
   OPTION memory='3m';
```

CREATE SYNCHRONIZATION USER statement [MobiLink]

Description

Use this statement in an Adaptive Server Anywhere remote database to create a synchronization user.

Syntax

CREATE SYNCHRONIZATION USER *ml_username*
[**TYPE** *protocol-type*]
[**ADDRESS** *protocol-options*]
[**OPTION** *option*=*value*, . . .]

ml_username: *identifier*

protocol-type: **tcpip** | **http** | **https** | **https_fips** | **ActiveSync**

protocol-options: *string*

value: *string* | *integer*

Parameters

ml_username A name identifying a remote database. This name must be unique.

☞ For more information about synchronization user names, see "About MobiLink users" on page 10.

TYPE clause This clause specifies the communication protocol to use for synchronization. The options are **tcpip**, **http**, **https**, **https_fips**, and **ActiveSync**. The default protocol is **tcpip**.

☞ For more information about communication protocols, see "CommunicationType (ctp) extended option" on page 111.

ADDRESS clause This clause specifies *protocol-options* in the form *keyword*=*value*, separated by semi-colons. Which settings you supply depends on the communication protocol you are using (TCPIP, HTTP, HTTPS, or ActiveSync).

☞ For a complete list of protocol options, see "CommunicationAddress (adr) extended option" on page 106.

OPTION clause The OPTION clause allows you to set extended options using *option*=*value* in a comma-separated list.

The values for each option cannot contain equal signs or semicolons. The database server accepts any option that you enter without checking for its validity. Therefore, if you misspell an option or enter an invalid value, no error message appears until you run the dbmlsync command to perform synchronization.

Options set for a synchronization user can be overridden in individual subscriptions or on the dbmlsync command line.

☞ For information about extended options, see “dbmlsync extended options” on page 105.

Description The *protocol-type*, *protocol-options*, and *options* can be set in several places.

☞ For information about how dbmlsync processes options that are specified in different locations, see “Priority order for extended options and connection parameters” on page 73.

Permissions Must have DBA authority.

Side effects Automatic commit.

See also

- ♦ “ALTER SYNCHRONIZATION USER statement [MobiLink]” on page 158
- ♦ “CREATE SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]” on page 162
- ♦ “CREATE PUBLICATION statement” on page 160
- ♦ “MobiLink Transport-Layer Security” [*MobiLink Administration Guide*, page 165]

Standards and compatibility

- ♦ **SQL/92** Vendor extension.
- ♦ **SQL/99** Vendor extension.
- ♦ **Sybase** Supported by Open Client/Open Server.

Examples The following example creates a user named SSinger, who synchronizes over TCP/IP with a server machine named mlserver.mycompany.com using the password Sam. The use of a password in the user definition is *not* secure.

```
CREATE SYNCHRONIZATION USER SSinger
TYPE http
ADDRESS 'host=mlserver.mycompany.com'
OPTION MobiLinkPwd='Sam'
```

The following creates a synchronization user called factory014 that will cause dbmlsync to hover and synchronize via Certicom-encrypted TCP/IP at a random time in every 8-hour interval. The randomness helps prevent performance degradation at the MobiLink server due to multiple simultaneous synchronizations:

```
CREATE SYNCHRONIZATION USER factory014
TYPE tcpip
ADDRESS 'host=mycompany.manufacturing.mobilink1;security=certico
         m_tls(certificate=mycompany_mobilink.crt;certificate_
         password=thepassword)'
OPTION Schedule='EVERY:08:00'
```

The following creates a synchronization user called sales5322 that will be used to synchronize with HTTP. In this example, the MobiLink synchronization server runs behind the corporate firewall, and synchronization requests are redirected to it using the Redirector (a reverse proxy to an NSAPI Web server).

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

DROP PUBLICATION statement

Description

Use this statement to drop a publication. In MobiLink a publication identifies synchronized data in a Adaptive Server Anywhere remote database. In SQL Remote, publications identify replicated data in both consolidated and remote databases.

Syntax

DROP PUBLICATION [*owner*.]*publication-name*

owner , *publication-name* : *identifier*

Usage

This statement is applicable only to MobiLink and SQL Remote.

Permissions

Must have DBA authority.

Side effects

Automatic commit. All subscriptions to the publication are dropped.

See also

- ♦ "ALTER PUBLICATION statement" on page 154
- ♦ "CREATE PUBLICATION statement" on page 160
- ♦ "sp_drop_publication procedure" [*SQL Remote User's Guide,* page 387]

Standards and compatibility

- ♦ **SQL/92** Vendor extension.
- ♦ **SQL/99** Vendor extension.

Example

The following statement drops the pub_contact publication.

```
DROP PUBLICATION pub_contact
```

DROP SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]

Description	Use this statement to drop a synchronization subscription within a MobiLink remote database or a MobiLink reference database. You can also use it to drop a default subscription, which contains default subscription values, for the specified publication.
Syntax	**DROP SYNCHRONIZATION SUBSCRIPTION** **TO** *publication-name* [**FOR** *ml_username*, . . .]
Parameters	**TO clause** Specify the name of a publication. **FOR clause** Specify one more MobiLink users. Omitting this clause drops the default subscription for the publication. MobiLink users subscribed to a publication inherit as defaults the values in a default publication.
Usage	Drop a synchronization subscription in a MobiLink remote or reference database.
Permissions	Must have DBA authority. Requires exclusive access to all tables referred to in the publication.
Side Effects	Automatic commit.
Standards and compatibility	♦ **SQL/92** Vendor extension. ♦ **SQL/99** Vendor extension.

Examples

Unsubscribe MobiLink user ml_user1 to the sales publication.

```
DROP SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
   FOR "ml_user1"
```

Drop the default subscription, which contains default subscription values, for the sales publication (by omitting the FOR clause).

```
DROP SYNCHRONIZATION SUBSCRIPTION
   TO sales_publication
```

DROP SYNCHRONIZATION USER statement [MobiLink]

Description	Use this statement to drop a synchronization user from a MobiLink remote database.
Syntax	**DROP SYNCHRONIZATION USER** *ml_username*, . . . *ml_username*: *identifier*
Usage	Drop one or more synchronization users from a MobiLink remote database.
Permissions	Must have DBA authority. Requires exclusive access to all tables referred to in the publication.
Side Effects	All subscriptions associated with the user are also deleted.
Standards and compatibility	♦ **SQL/92** Vendor extension. ♦ **SQL/99** Vendor extension.
Example	Remove MobiLink user ml_user1 from the database.

```
DROP SYNCHRONIZATION USER ml_user1
```

START SYNCHRONIZATION DELETE statement [MobiLink]

Description

Use this statement to restart logging of deletes for MobiLink synchronization.

Syntax

START SYNCHRONIZATION DELETE

Usage

Ordinarily, Adaptive Server Anywhere and UltraLite automatically log any changes made to tables or columns that are part of a synchronization, and upload these changes to the consolidated database during the next synchronization. You can temporarily suspend automatic logging of delete operations using the STOP SYNCHRONIZATION DELETE statement. The START SYNCHRONIZATION DELETE statement allows you to restart the automatic logging.

When a STOP SYNCHRONIZATION DELETE statement is executed, none of the delete operations executed on that connection will be synchronized. The effect continues until a START SYNCHRONIZATION DELETE statement is executed. Repeating STOP SYNCHRONIZATION DELETE has no additional effect.

A single START SYNCHRONIZATION DELETE statement restarts the logging, regardless of the number of STOP SYNCHRONIZATION DELETE statements preceding it.

Do not use START SYNCHRONIZATION DELETE if your application does not synchronize data.

Permissions

Must have DBA authority.

Side effects

None.

See also

♦ "STOP SYNCHRONIZATION DELETE statement [MobiLink]" on page 172
♦ "StartSynchronizationDelete method" [*UltraLite C/C++ User's Guide*, page 317]

Standards and compatibility

♦ **SQL/92** Vendor extension.

♦ **SQL/99** Vendor extension.

♦ **Sybase** Not applicable.

Example

The following sequence of SQL statements illustrates how to use START SYNCHRONIZATION DELETE and STOP SYNCHRONIZATION DELETE.

```
-- Prevent deletes from being sent
-- to the consolidated database
STOP SYNCHRONIZATION DELETE;

-- Remove all records older than 1 month
-- from the remote database,
-- NOT the consolidated database
DELETE FROM PROPOSAL
WHERE last_modified < months( CURRENT TIMESTAMP, -1 )

-- Re-enable all deletes to be sent
-- to the consolidated database
-- DO NOT FORGET to start this
START SYNCHRONIZATION DELETE;

-- Commit the entire operation,
-- otherwise rollback everything
-- including the stopping of the deletes
commit;
```

STOP SYNCHRONIZATION DELETE statement [MobiLink]

Description

Use this statement to temporarily stop logging of deletes for MobiLink synchronization.

Syntax

STOP SYNCHRONIZATION DELETE

Usage

Ordinarily, Adaptive Server Anywhere and UltraLite remote databases automatically log any changes made to tables or columns that are included in a synchronization, and then upload these changes to the consolidated database during the next synchronization. This statement allows you to temporarily suspend logging of changes to an Adaptive Server Anywhere or UltraLite remote database.

When a STOP SYNCHRONIZATION DELETE statement is executed, none of the subsequent delete operations executed on that connection will be synchronized. The effect continues until a START SYNCHRONIZATION DELETE statement is executed.

Repeating STOP SYNCHRONIZATION DELETE has no additional effect. A single START SYNCHRONIZATION DELETE statement restarts the logging, regardless of the number of STOP SYNCHRONIZATION DELETE statements preceding it.

This command can be useful to make corrections to a remote database, but should be used with caution as it effectively disables MobiLink synchronization.

Do not use STOP SYNCHRONIZATION DELETE if your application does not synchronize data.

Permissions

Must have DBA authority.

Side Effects

None.

See also

- ♦ “StartSynchronizationDelete method” [*UltraLite C/C++ User’s Guide*, page 317]
- ♦ “StopSynchronizationDelete method” [*UltraLite C/C++ User’s Guide*, page 317]
- ♦ “START SYNCHRONIZATION DELETE statement [MobiLink]” on page 170

Standards and compatibility

- ♦ **SQL/92** Vendor extension.
- ♦ **SQL/99** Vendor extension.
- ♦ **Sybase** Not applicable.

Example

☞ For an example, see "START SYNCHRONIZATION DELETE statement [MobiLink]" on page 170.

CHAPTER 8

Dbmlsync Client Event Hooks

About this chapter

Contents

Customizing the client synchronization process

The Adaptive Server Anywhere synchronization client, dbmlsync, provides a 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 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 synchronization events, schedule synchronizations based on logical events or time and customize synchronization behavior.

In addition, you can use event hooks to synchronize subsets of data that cannot be easily defined in a publication. For example, you can synchronize data in a temporary table by writing one event hook procedure to copy data from the temporary table to a permanent table prior to the synchronization and another to copy the data back afterwards.

Caution
The integrity of the synchronization process relies on a sequence of built-in transactions. Thus, you must not perform an implicit or explicit commit or rollback within your event-hook procedures.

Synchronization event hook sequence

The following pseudo-code shows the available events and the point at which each is called during the synchronization process. For example, sp_hook_dbmlsync_abort is the first event hook to be invoked.

Each event makes particular parameter values available, which you can use when you implement the procedure. In some cases, you can modify the value to return a new value; others are read-only. These parameters are not stored procedure arguments. No arguments are passed to any of the event-hook stored procedures. Instead, arguments are exchanged by reading and modifying rows in the #hook_dict table.

For example, the sp_hook_dbmlsync_begin procedure has a parameter, which is the user name that the application supplied in the synchronization call. You can retrieve this value from the #hook_dict table.

Although the sequence has similarities to the event sequence at the MobiLink synchronization 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.

Any *_end hook will be called if the corresponding *_begin hook is called and completed successfully. A *_begin hook is considered to have run successfully if it was not implemented when it would have been called.

☞ For more information about upload options, see "-tu option" on page 147 and "Increment (inc) extended option" on page 121.

```
sp_hook_dbmlsync_abort
sp_hook_dbmlsync_set_extended_options
loop until return codes direct otherwise (
   sp_hook_dbmlsync_abort
   sp_hook_dbmlsync_delay
)
sp_hook_dbmlsync_abort
// start synchronization
sp_hook_dbmlsync_begin
// upload events
for each upload segment
// a normal synchronization has one upload segment
// a transaction-level upload has one segment per transaction
// an incremental upload has one segment per upload piece
 sp_hook_dbmlsync_logscan_begin
 sp_hook_dbmlsync_logscan_end
 sp_hook_dbmlsync_upload_begin
 sp_hook_dbmlsync_upload_end
next upload event

// 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_return_code
sp_hook_dbmlsync_log_rescan
```

Error handling

In addition, the following event-hook procedures are available for error handling.

```
sp_hook_dbmlsync_all_error
sp_hook_dbmlsync_communication_error
sp_hook_dbmlsync_connect_failed
sp_hook_dbmlsync_download_com_error
sp_hook_dbmlsync_download_SQL_error
sp_hook_dbmlsync_download_fatal_SQL_error
sp_hook_dbmlsync_download_ri_violation
sp_hook_dbmlsync_download_log_ri_violation
sp_hook_dbmlsync_misc_error
sp_hook_dbmlsync_sql_error
```

Once implemented, each procedure is automatically executed whenever an error of the named type occurs.

Using event-hook procedures

This section describes some considerations for designing and using event-hook procedures.

Notes

- Do not perform any COMMIT or ROLLBACK operations in event-hook procedures. The procedures are executed on the same connection as the synchronization, and a COMMIT or ROLLBACK may interfere with synchronization.
- Do not define more than one hook with the same name. If more than one hook with the same name is created (say by different users), then which hook is called is undefined.
- Hook procedures must be created by a user with DBA authority.
- If a *_begin hook executes successfully, the corresponding *_end hook is called regardless of any error that occurs afterwards. If the *_begin hook is not defined, but you have defined an *_end hook, then the *_end hook is called unless an error occurs prior the point in time where the *_begin hook would normally be called.

#hook_dict table

Immediately before a hook is called, dbmlsync creates the #hook_dict table in the remote database, using the following CREATE statement. The # before the table name means that the table is temporary.

```
CREATE TABLE #hook_dict(
name VARCHAR(128) NOT NULL UNIQUE,
value VARCHAR(255) NOT NULL)
```

dbmlsync 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.

For example, for the following dbmlsync command line,

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

when the sp_hook_dbmlsync_abort hook is called, the #hook_dict table will contain the following rows:

Name	Value
publication_0	`pub1`
publication_1	`pub2`
MobiLink user	`MyUser`
Abort synchronization	`false`

Your abort hook can retrieve values from the #hook_dict table and use them to customize behavior. For example, to retrieve the MobiLink user you would use a SELECT statement like this:

```
SELECT value
FROM #hook_dict
WHERE name = ’MobiLink user’
```

In/out parameters can be updated by your hook to modify the behavior of dbmlsync. For example, your hook could instruct dbmlsync to abort synchronization by updating the abort synchronization row of the table using a statement like this:

```
UPDATE #hook_dict
SET value=’true’
WHERE name=’abort synchronization’
```

The description of each hook lists the rows in the #hook_dict table.

Event-hook procedure owner

The event-hook connection calls the stored procedures without qualifying them by owner. The stored procedures must therefore be owned by one of the following:

- The user name employed on the dbmlsync connection (typically a user with REMOTE DBA authority).

- A group ID of which the dbmlsync user is a member.

Connections for event-hook procedures

Each event-hook procedure is executed on the same connection as the synchronization itself. The following are exceptions:

- sp_hook_dbmlsync_all_error
- sp_hook_dbmlsync_communication_error
- sp_hook_dbmlsync_download_com_error
- sp_hook_dbmlsync_download_fatal_sql_error
- sp_hook_dbmlsync_download_log_ri_violation
- sp_hook_dbmlsync_misc_error
- sp_hook_dbmlsync_sql_error

These procedures are called before a synchronization fails. On failure, synchronization actions are rolled back. By operating 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.

Event arguments

Each hook receives parameter values. In some cases, you can modify the value to return a new value; others are read-only.

These parameters are exchanged by reading and modifying rows in the #hook_dict table, which is defined as follows.

```
CREATE TABLE #hook_dict (
   name   VARCHAR( 128 ) NOT NULL UNIQUE,
   value   VARCHAR( 255 ) NOT NULL
)
```

Each row in the table contains the value for one parameter.

Before calling any of the stored procedures, dbmlsync creates the #hook_dict table, and adds the parameters for that event. Procedures can read the values by selecting from this table.

Some parameters can be used to pass values back to dbmlsync from the hook. The hook passes values back by updating the #hook_dict table.

☞ For a list of the parameter values supplied at each event, see "Dbmlsync Client Event Hooks" on page 175.

Examples

The following examples illustrate how to retrieve and set values in the #hook_dict table.

The following sample sp_hook_dbmlsync_delay procedure illustrates the use of the #hook_dict table to pass arguments. The procedure allows synchronization only outside a scheduled down time of the MobiLink system between 18:00 and 19:00.

```
CREATE PROCEDURE sp_hook_dbmlsync_delay()
BEGIN
   DECLARE delay_val integer;
 SET delay_val=DATEDIFF(
   second, CURRENT TIME, ’19:00’);
 IF (delay_val>0 AND
     delay_val<3600)
 THEN
 UPDATE #hook_dict SET value=delay_val
   WHERE name=’delay duration’;
 END IF;
END
```

The following procedure is executed in the remote database at the beginning of synchronization. It retrieves the current MobiLink user name, one of the parameters available for the sp_hook_dbmlsync_begin event, and displays it on the console.

```
CREATE PROCEDURE sp_hook_dbmlsync_begin()
BEGIN
   DECLARE syncdef VARCHAR(150);
   SELECT ’>>>syncdef = ’ || value INTO syncdef
      FROM #hook_dict
      WHERE name =’MobiLink user name’;
   MESSAGE syncdef TYPE INFO TO CONSOLE;
END
```

Ignoring errors in event-hook procedures

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

For more information, see “IgnoreHookErrors (eh) extended option” on page 120.

sp_hook_dbmlsync_abort

Function

Use this stored procedure to cancel the synchronization process.

Rows in #hook_dict table

Name	Values	Description
abort synchronization (in|out)	**true** | **false**	If you set the abort synchronization row of the #hook_dict table to **true**, then dbmlsync terminates immediately after the event.
publication_*n* (in)	publication name	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
exit code (in|out)	number	When abort synchronization is set to TRUE, you can use this value to set the return 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.

Description

If a procedure of this name exists, it is called at dbmlsync startup, and then again after each synchronization delay that is caused by the sp_hook_dbmlsync_delay hook.

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

Actions of this procedure are committed immediately after execution.

See also

- "Synchronization event hook sequence" on page 177
- "sp_hook_dbmlsync_process_return_code" on page 221

Examples

The following procedure prevents synchronization during a scheduled maintenance hour between 19:00 and 20:00 each day.

```
create procedure sp_hook_dbmlsync_abort()
begin
  declare down_time_start time;
  declare is_down_time varchar(128);
  set down_time_start='19:00';
  if abs( datediff( hour,down_time_start,now(*) ) ) < 1
  then
    set is_down_time='true';
  else
    set is_down_time='false';
  end if;
  UPDATE #hook_dict
  SET value = is_down_time
  WHERE name = 'abort synchronization'
end
```

Suppose you have an abort hook that may abort synchronization for one of two reasons. One of the reasons indicates normal completion of synchronization, so you want dbmlsync to have an exit code of 0. The other reason indicates an error condition, so you want dbmlsync to have a non-zero exit code. You could achieve this with an sp_hook_dbmlsync_abort hook defined as follows.

```
BEGIN
   IF [condition that defines the normal abort case] THEN
      UPDATE #hook_dict SET value =  '0'
   WHERE name = 'exit code';
      UPDATE #hook_dict SET value =  'TRUE'
   WHERE name = 'abort synchronization';
   ELSEIF [condition that defines the error abort case] THEN
      UPDATE #hook_dict SET value =  '1'
   WHERE name = 'exit code';
      UPDATE #hook_dict SET value =  'TRUE'
   WHERE name = 'abort synchronization';
   END IF;
END;
```

sp_hook_dbmlsync_begin

Function

Use this stored procedure to add custom actions at the beginning of the synchronization process.

Rows in #hook_dict table

Name	Values	Description
publication_*n* (in)	publication name	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.

Description

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

Actions of this procedure are committed immediately after execution.

See also

- "Synchronization event hook sequence" on page 177

Examples

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
 "event_id"           integer NOT NULL DEFAULT autoincrement ,
   "event_name"       varchar(128) NOT NULL ,
   "ml_user"            varchar(128) NULL ,
   "event_time"         timestamp NULL,
   "table_name"         varchar(128) NULL ,
   "upsert_count"       varchar(128) NULL ,
   "delete_count"       varchar(128) NULL ,
   "exit_code"          integer NULL ,
   "status_retval"      varchar(128) NULL ,
   "pubs"                 varchar(128) NULL ,
   "sync_descr "        varchar(128) NULL ,
    PRIMARY KEY ("event_id"),
)
```

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

```
CREATE PROCEDURE sp_hook_dbmlsync_begin ()
BEGIN

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

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

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

sp_hook_dbmlsync_connect_failed

Function

Use this stored procedure to retry failed attempts to connect to the MobiLink synchronization server using a different communication type or address.

Rows in #hook_dict table

Name	Values	Description
publication_*n* (in)	publication name	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.
connection address (in|out)	connection address	When the hook is invoked, this is the address used in the most recent failed communication attempt. You can set this value to a new connection address that you want to try. If retry is set to true, this value is used for the next communication attempt.
connection type (in|out)	**tcpip** | **http** | **https** | **ActiveSync** | **ecc_tls** | **rsa_tls**	When the hook is invoked, this is the communication type that was used in the most recent failed communication attempt. You can set this value to a new connection type that you want to try. If retry is set to true, this value is used for the next communication attempt.
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.

Name	Values	Description
allow remote ahead (in|out)	**true** | **false**	This is true only if dbmlsync was started with the -ra option. You can use this row to read or change the -ra option for the current synchronization only. For more information, see "-r option" on page 146.
allow remote behind (in|out)	**true** | **false**	This is true only if dbmlsync was started with the -rb option. You can use this row to read or change the -rb option for the current synchronization only. For more information, see "-r option" on page 146.
retry (in|out)	**true** | **false**	Set this value to true if you want to retry a failed connection attempt. The default is FALSE.

Description

If a procedure of this name exists, it is called if dbmlsync fails while attempting to connect to the MobiLink synchronization server.

This hook only applies to connection attempts to the MobiLink synchronization server, not the database.

When a progress offset mismatch occurs, dbmlsync disconnects from the MobiLink synchronization server and reconnects later. In this kind of reconnection, this hook is not called, and failure to reconnect causes the synchronization to fail.

Actions of this procedure are committed immediately after execution.

Examples

This example uses the sp_hook_dbmlsync_connect_failed hook to retry the connection up to five times.

```
CREATE PROCEDURE sp_hook_dbmlsync_ml_connect_failed ()
BEGIN
    DECLARE idx integer;
    DECLARE buf varchar( 128 );

    SELECT value
      INTO buf
      FROM #hook_dict
      WHERE name = 'user data';

    IF buf = '' THEN
      SELECT 1 INTO idx;
    ELSE
      SELECT CONVERT( integer, value ) + 1
        INTO idx
        FROM #hook_dict
        WHERE name = 'user data';
    END IF;

  IF idx <= 5 THEN
        UPDATE #hook_dict
        SET value = CONVERT( varchar(128), idx )
        WHERE name = 'user data';

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

The next example uses a table containing connection information. When an attempt to connect fails, the hook tries the next server in the list.

```
CREATE TABLE conn_list (
    label   integer primary key,
    addr  varchar( 128 ),
    type  varchar( 64 )
);
INSERT INTO conn_list
 VALUES ( 1, 'host=server1;port=91', 'tcpip' );
INSERT INTO conn_list
 VALUES ( 2, 'host=server2;port=92', 'http' );
INSERT INTO conn_list
 VALUES ( 3, 'host=server3;port=93', 'tcpip' );
COMMIT;

CREATE PROCEDURE sp_hook_dbmlsync_ml_connect_failed ()
BEGIN
 DECLARE idx integer;
 DECLARE cnt integer;
 DECLARE buf varchar( 128 );

 SELECT value
  INTO buf
  FROM #hook_dict
  WHERE name = 'user data';

        IF buf = '' THEN
   SELECT 1 INTO idx;
 ELSE
    SELECT CONVERT( integer, value ) + 1
   INTO idx
     FROM #hook_dict
   WHERE name = 'user data';
   END IF;

    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 = CONVERT( varchar(128), idx )
        WHERE name = 'user data';

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

sp_hook_dbmlsync_delay

Function — Use this stored procedure to control when synchronization takes place.

Rows in #hook_dict table

Name	Values	Description
delay duration (in|out)	number of seconds	If the procedure sets the **delay duration** value to zero, then dbmlsync synchronization proceeds. A non-zero **delay_duration** value specifies the number of seconds before the delay hook is called again.
maximum accumulated delay (in|out)	number of seconds	The maximum accumulated delay specifies the maximum number of seconds delay before each synchronization. Dbmlsync keeps track of the total delay created by all calls to the delay hook since the last synchronization. If no synchronization has occurred since dbmlsync started running, the total delay is calculated from the time dbmlsync started up. When the total delay exceeds the value of maximum accumulated delay, synchronization begins without any further calls to the delay hook.
publication_*n* (in)	publication name	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.

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

Actions of this procedure are committed immediately after execution.

See also

♦ "Synchronization event hook sequence" on page 177

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.

```
-- change status of high priority row
 UPDATE OrdersTable
  SET priority = 'high-processed'
  WHERE priority = 'high';
```

☞ For more information about sp_hook_dbmlsync_end, see "sp_hook_dbmlsync_end" on page 212.

sp_hook_dbmlsync_download_begin

Function

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	Values	Description
publication_*n* (in)	publication name	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.

Description

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 stream is committed or rolled back.

See also

- "Synchronization event hook sequence" on page 177

Example

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
 "event_id"            integer NOT NULL DEFAULT autoincrement ,
   "event_name"        varchar(128) NOT NULL ,
   "ml_user"             varchar(128) NULL ,
   "event_time"          timestamp NULL,
   "table_name"          varchar(128) NULL ,
   "upsert_count"        varchar(128) NULL ,
   "delete_count"        varchar(128) NULL ,
   "exit_code"           integer NULL ,
   "status_retval"       varchar(128) NULL ,
   "pubs"                  varchar(128) NULL ,
   "sync_descr "         varchar(128) NULL ,
    PRIMARY KEY ("event_id"),
)
```

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

```
CREATE PROCEDURE sp_hook_dbmlsync_begin ()
BEGIN

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

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

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

sp_hook_dbmlsync_download_com_error

Function

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

Rows in #hook_dict table

Name	Values	Description
table name (in)	table name	The table to which operations were being applied when the error occurred. The value is an empty string if dbmlsync is unable to identify the table.
publication_*n* (in)	publication name	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.

Description

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

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

By default on Windows CE devices, synchronization tables are locked in exclusive mode, which means that this hook cannot successfully execute if it requires access to any of the synchronization tables. It also cannot execute if it needs to access synchronization tables and you set the dbmlsync extended option LockTables to EXCLUSIVE. For more information, see "LockTables (lt) extended option" on page 122.

Actions of this procedure are committed immediately after execution.

See also

♦ "Synchronization event hook sequence" on page 177

Examples

Assume you use the following table to log communication errors.

```
CREATE TABLE SyncLogComErrorTable
(
   " user_name "    varchar(255) NOT NULL ,
   " event_time "   timestamp NOT NULL ,
)
```

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

```
CREATE PROCEDURE sp_hook_dbmlsync_download_com_error ()
BEGIN
 INSERT INTO SyncLogComErrorTable (user_name, event_time)
  SELECT #hook_dict.value, current timestamp
  FROM #hook_dict
  WHERE name = 'MobiLink user';
END;
```

sp_hook_dbmlsync_download_end

Function

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	Values	Description
publication_*n* (in)	publication name	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.

Description

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 stream is committed or rolled back.

See also

♦ "Synchronization event hook sequence" on page 177

Examples

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
 "event_id"            integer NOT NULL DEFAULT autoincrement ,
   "event_name"        varchar(128) NOT NULL ,
   "ml_user"             varchar(128) NULL ,
   "event_time"          timestamp NULL,
   "table_name"          varchar(128) NULL ,
   "upsert_count"        varchar(128) NULL ,
   "delete_count"        varchar(128) NULL ,
   "exit_code"           integer NULL ,
   "status_retval"       varchar(128) NULL ,
   "pubs"                  varchar(128) NULL ,
   "sync_descr "         varchar(128) NULL ,
    PRIMARY KEY ("event_id"),
)
```

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

```
CREATE PROCEDURE sp_hook_dbmlsync_begin ()
BEGIN

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

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

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

sp_hook_dbmlsync_download_fatal_sql_error

Function

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

Rows in #hook_dict table

Name	Values	Description
table name (in)	table name	The table to which operations were being applied when the error occurred. The value is an empty string if dbmlsync is unable to identify the table.
SQL error code (in)	SQL error code	Identifies the SQL error code returned by the database when the operation failed.
publication_*n* (in)	publication name	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
script version (in)	script version name	The MobiLink script version to be used for the synchronization.

Description

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

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

By default on Windows CE devices, synchronization tables are locked in exclusive mode, which means that this hook cannot successfully execute if it requires access to any of the synchronization tables. It also cannot execute if it needs to access synchronization tables and you set the dbmlsync extended option LockTables to EXCLUSIVE. For more information, see “LockTables (lt) extended option” on page 122.

Actions of this procedure are committed immediately after execution.

See also

- “Synchronization event hook sequence” on page 177
- “sp_hook_dbmlsync_download_sql_error” on page 206

Examples

Assume you use the following table to log SQL errors.

```
CREATE TABLE "DBA"."SyncLogComErrorTable"
(
   " error_code "       varchar(255) NOT NULL ,
   " event_time "       timestamp NOT NULL ,
)
```

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

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

sp_hook_dbmlsync_download_log_ri_violation

Function

Logs referential integrity violations in the download process.

Rows in #hook_dict table

Name	Values	Description
Publication_*n* (in)	publication name	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
Foreign key table (in)	table name	The table containing the foreign key column for which the hook is being called.
Primary key table (in)	table name	The table referenced by the foreign key for which the hook is being called.
Role name (in)	role name	The role name of the foreign key for which the hook is being called.
Script version (in)	script version name	The MobiLink script version to be used for the synchronization.

Description

A download RI violation occurs when rows in the download stream violate foreign key relationships on the remote database. This hook allows you to log RI violations as they occur so that you can investigate their cause later.

After the download is complete, but before it is committed, dbmlsync checks for RI violations. If it finds any, it identifies a foreign key that has an RI violation and calls sp_hook_dbmlsync_download_log_ri_violation (if it is implemented). It then calls sp_hook_dbmlsync_download_ri_conflict (if it is implemented). If there is still a conflict, dbmlsync deletes the rows that violate the foreign key constraint. This process is repeated for remaining foreign keys that have RI violations.

This hook is called only when there are RI violations involving tables that are currently being synchronized. If there are RI violations involving tables that are not being synchronized, this hook is not called and the synchronization fails.

This hook is called on a separate connection from the one that dbmlsync uses for the download. The connection used by the hook has an isolation level of 0 so that the hook can see the rows that have been applied from the download stream that are not yet committed. The actions of the hook are committed immediately after it completes so that changes made by this hook will be preserved regardless of whether the download stream is committed or rolled back.

By default on Windows CE devices, synchronization tables are locked in exclusive mode, which means that this hook cannot successfully execute if it requires access to any of the synchronization tables. It also cannot execute if it needs to access synchronization tables and you set the dbmlsync extended option LockTables to EXCLUSIVE. For more information, see "LockTables (lt) extended option" on page 122.

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

See also

- "sp_hook_dbmlsync_download_ri_violation" on page 204
- "Synchronization event hook sequence" on page 177

Examples

Assume you use the following table to log referential integrity violations.

```
CREATE TABLE "DBA"." LogRIErrorTable "
(
 "id"     int PRIMARY KEY NOT NULL,
 "name"     varchar(255) NOT NULL,
 "value"  timestamp NOT NULL
)
```

The following example logs the foreign key table name, primary key table name, and other synchronization information when a referential integrity violation is detected on the remote database. The information is stored in LogRIErrorTable on the remote database.

```
CREATE PROCEDURE sp_hook_dbmlsync_download_log_ri_violation()
BEGIN
 --
 -- get the next id for the LogRIErrorTable
 --
   DECLARE nextid int;

 SELECT max(id)
  INTO nextid
  FROM LogRIErrorTable;

 IF nextid is NULL THEN
  SET nextid = 0
 ELSE
  SET nextid = nextid +1
 END IF;

 INSERT INTO LogRIErrorTable(id,name, value )
  SELECT nextid,#hook_dict.name, #hook_dict.value
  FROM #hook_dict;
END;
```

sp_hook_dbmlsync_download_ri_violation

Function

Allows you to resolve referential integrity violations in the download process.

Rows in #hook_dict table

Name	Values	Description
Publication_*n* (in)	publication name	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
Foreign key table (in)	table name	The table containing the foreign key column for which the hook is being called.
Primary key table (in)	table name	The table referenced by the foreign key for which the hook is being called.
Role name (in)	role name	The role name of the foreign key for which the hook is being called.
Script version (in)	script version name	The MobiLink script version to be used for the synchronization.

Description

A download RI violation occurs when rows in the download stream violate foreign key relationships on the remote database. This hook allows you to attempt to resolve RI violations before dbmlsync deletes the rows that are causing the conflict.

After the download is complete, but before it is committed, dbmlsync checks for RI violations. If it finds any, it identifies a foreign key that has an RI violation and calls sp_hook_dbmlsync_download_log_ri_violation (if it is implemented). It then calls sp_hook_dbmlsync_download_ri_conflict (if it is implemented). If there is still a conflict, dbmlsync deletes the rows. This process is repeated for remaining foreign keys that have RI violations.

This hook is called only when there are RI violations involving tables that are currently being synchronized. If there are RI violations involving tables that are not being synchronized, this hook is not called and the synchronization fails.

This hook is called on the same connection that dbmlsync uses for the download. This hook should not contain any explicit or implicit commits, because they may lead to inconsistent data in the database. The actions of this hook are committed or rolled back when the download stream is committed or rolled back.

Unlike other hook actions, the operations performed during this hook are not uploaded during the next synchronization.

See also

- "sp_hook_dbmlsync_download_log_ri_violation" on page 201

Example

This example uses the Department and Employee tables shown below:

```
CREATE TABLE Department(
 "department_id"   integer primary key
);

CREATE TABLE Employee(
  "employee_id"      integer primary key,
  "department_id"   integer,
  foreign key EMPLOYEE_FK1 (department_id) references Department
)
```

The following sp_hook_dbmlsync_download_ri_violation definition cleans up referential integrity violations between the Department and Employee tables. It verifies the role name for the foreign key and inserts missing department_id values into the Department table.

```
CREATE PROCEDURE sp_hook_dbmlsync_download_ri_violation()
BEGIN

IF EXISTS (SELECT * FROM #hook_dict WHERE name = 'role name'
 AND value = 'EMPLOYEE_FK1') THEN

 -- update the Department table with missing department_id
         values
 INSERT INTO Department
  SELECT distinct department_id FROM Employee
  WHERE department_id NOT IN (SELECT department_id FROM
         Department)

END IF;
```

sp_hook_dbmlsync_download_sql_error

Function

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

Rows in #hook_dict table

Name	Values	Description
Table name (in)	table name	The table to which operations were being applied when the error occurred. The value is an empty string if dbmlsync is unable to identify the table.
Continue (in|out)	**true** | **false**	Indicates whether the error should be ignored and synchronization should continue. This parameter should be set to **false** to call the sp_hook_dbmlsync_download_-fatal_sql_error hook and stop synchronization. If you set this parameter to **true**, dbmlsync ignores the error and continues with synchronization, which may result in data loss.
SQL error code (in)	SQL error code	Identifies the SQL error code returned by the database when the operation failed.
Publication_*n* (in)	publication name	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
Script version (in)	script version name	The MobiLink script version to be used for the synchronization.

Description

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

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

Actions of this procedure are committed or rolled back when the download stream is committed or rolled back.

See also

- "Synchronization event hook sequence" on page 177
- "sp_hook_dbmlsync_download_fatal_sql_error" on page 199

Examples

Assume you use the following table to log SQL errors.

```
CREATE TABLE "DBA"." LogSQLErrorTable "
(
 "id"     int PRIMARY KEY NOT NULL,
 "name"     varchar(255) NOT NULL,
 "value"  timestamp NOT NULL
)
```

The following example logs the SQL error code, table name, and other synchronization information when a database error occurs during the download phase of synchronization.

```
CREATE PROCEDURE sp_hook_dbmlsync_download_sql_error()
BEGIN
 --
 -- get the next LogSQLErrorTable id
 --
   DECLARE nextid int;

 SELECT max(id)
  INTO nextid
  FROM LogSQLErrorTable;

 IF nextid is NULL THEN
  SET nextid = 0
 ELSE
  SET nextid = nextid +1
 END IF;

 INSERT INTO LogSQLErrorTable(id,name, value )
  SELECT nextid,#hook_dict.name, #hook_dict.value
  FROM #hook_dict;
END;
```

sp_hook_dbmlsync_download_table_begin

Function

Use this stored procedure to add custom actions immediately before each table is downloaded.

Rows in #hook_dict table

Name	Values	Description
Table name (in)	table name	The table to which operations are about to be applied.
Publication_*n* (in)	publication name	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
Script version (in)	script version name	The MobiLink script version to be used for the synchronization.

Description

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 stream is committed or rolled back.

See also

♦ "Synchronization event hook sequence" on page 177

Examples

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
 "event_id"             integer NOT NULL DEFAULT autoincrement ,
   "event_name"         varchar(128) NOT NULL ,
   "ml_user"              varchar(128) NULL ,
   "event_time"           timestamp NULL,
   "table_name"           varchar(128) NULL ,
   "upsert_count"         varchar(128) NULL ,
   "delete_count"         varchar(128) NULL ,
   "exit_code"            integer NULL ,
   "status_retval"        varchar(128) NULL ,
   "pubs"                   varchar(128) NULL ,
   "sync_descr "          varchar(128) NULL ,
    PRIMARY KEY ("event_id"),
)
```

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

```
CREATE PROCEDURE sp_hook_dbmlsync_download_table_begin()
BEGIN
    declare tbl varchar(255);

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

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

sp_hook_dbmlsync_download_table_end

Function

Use this stored procedure to add custom actions immediately after each table is downloaded.

Rows in #hook_dict table

Name	**Values**	**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 stream.
Upsert count (in)	number of rows	The number of rows in this table updated or inserted by the download stream.
Publication_*n* (in)	publication name	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
Script version (in)	script version name	The MobiLink script version to be used for the synchronization.

Description

If a procedure of this name exists, it is called immediately after all operations in the download stream for a table have been applied.

Actions of this procedure are committed or rolled back when the download stream is committed or rolled back.

See also

- “Synchronization event hook sequence” on page 177

Examples

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
 "event_id"            integer NOT NULL DEFAULT autoincrement ,
   "event_name"        varchar(128) NOT NULL ,
   "ml_user"             varchar(128) NULL ,
   "event_time"          timestamp NULL,
   "table_name"          varchar(128) NULL ,
   "upsert_count"        varchar(128) NULL ,
   "delete_count"        varchar(128) NULL ,
   "exit_code"           integer NULL ,
   "status_retval"       varchar(128) NULL ,
   "pubs"                  varchar(128) NULL ,
   "sync_descr "         varchar(128) NULL ,
    PRIMARY KEY ("event_id"),
)
```

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

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

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

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

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

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

sp_hook_dbmlsync_end

Function

Use this stored procedure to add custom actions immediately before synchronization is complete.

Rows in #hook_dict table

Name	Values	Description
Restart (out)	**sync** | **download** | **none**	If set to **sync**, then dbmlsync retries the synchronization it just completed. The value **sync** replaces **true**, which is identical but is deprecated. If set to **none** (the default), then dbmlsync shuts down or restarts according to its command line arguments. The value **none** replaces **false**, which is identical but is deprecated. If set to **download** and the restartable download parameter is **true**, then dbmlsync attempts to restart the download that just failed.
Exit code (in)	number	If set to anything other than zero (the default), this represents a synchronization error.
Publication_*n* (in)	publication name	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.

Name	Values	Description
Upload status (in)	**committed** | **failed**	Specifies the status returned by the MobiLink synchronization server when dbmlsync attempted to verify receipt of the upload stream. **committed** The upload stream was received by the MobiLink synchronization server, and committed. **failed** The MobiLink synchronization server did not commit the upload stream.
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.

Description

If a procedure of this name exists, it is called as the last event during synchronization.

Actions of this procedure are committed immediately after execution.

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

See also

- "Customizing the client synchronization process" on page 177
- "Synchronization event hook sequence" on page 177
- "Resuming failed downloads" [*MobiLink Administration Guide*, page 74]

Examples

In the following example the download is manually restarted if the download for the current synchronization failed and can be restarted.

```
CREATE PROCEDURE sp_hook_dbmlsync_end()
BEGIN
  -- Restart the download if the download for the current sync
  --  failed and can be restarted
    IF exists(SELECT * FROM #hook_dict
    WHERE name = 'restartable download' AND value='true')
      THEN
   UPDATE #hook_dict SET value ='download' WHERE name='restart';
  END IF;
END
```

sp_hook_dbmlsync_log_rescan

Function

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

Rows in #hook_dict table

Name	Values	Description
Publication_*n* (in)	publication name	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
Discarded storage (in)	number	The number of bytes of discarded memory after the last synchronization.
Rescan (in|out)	**true** | **false**	If set to True by the hook, dbmlsync performs a complete rescan before the next synchronization. On entry, this value is set to False.
Script version (in)	script version name	The MobiLink script version to be used for the synchronization.

Description

When more than one -n option is specified at the command line, dbmlsync may experience fragmentation which results in discarded memory. 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_end hook.

See also

- "HoverRescanThreshold (hrt) extended option" on page 119

Examples

The following example sets the rescan field in the #hook_dict table to TRUE if the discarded storage is greater than 1000 bytes.

```
CREATE PROCEDURE sp_hook_dbmlsync_log_rescan ()
BEGIN

 IF exists(SELECT * FROM #hook_dict
  WHERE name = 'Discarded storage' AND value>1000)
 THEN
  UPDATE #hook_dict SET value ='true' WHERE name='Rescan';
 END IF;

END
```

sp_hook_dbmlsync_logscan_begin

Function

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

Rows in #hook_dict table

Name	Values	Description
Starting log offset_*n* (in)	number	The log offset value where scanning is to begin. There is one value for each publication being uploaded. The numbering of *n* starts at zero.
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 synchronization server and dbmlsync have different information about where the scanning should begin.
Publication_*n* (in)	publication name	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
Script version (in)	script version name	The MobiLink script version to be used for the synchronization.

Description

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

Actions of this procedure are committed immediately after execution.

See also

♦ “Synchronization event hook sequence” on page 177

Examples

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
 "event_id"           integer NOT NULL DEFAULT autoincrement ,
   "event_name"       varchar(128) NOT NULL ,
   "ml_user"            varchar(128) NULL ,
   "event_time"         timestamp NULL,
   "table_name"         varchar(128) NULL ,
   "upsert_count"       varchar(128) NULL ,
   "delete_count"       varchar(128) NULL ,
   "exit_code"          integer NULL ,
   "status_retval"      varchar(128) NULL ,
   "pubs"                 varchar(128) NULL ,
   "sync_descr "        varchar(128) NULL ,
    PRIMARY KEY ("event_id"),
)
```

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

```
CREATE PROCEDURE sp_hook_dbmlsync_logscan_begin ()
BEGIN
 -- log the synchronization event
 INSERT INTO SyncLog (event_nam
e, ml_user,event_time)
  SELECT ’logscan_begin’, #hook_dict.value, current timestamp
  FROM #hook_dict
  WHERE name = ’MobiLink user’ ;
END
```

sp_hook_dbmlsync_logscan_end

Function

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

Rows in #hook_dict table

Name	Values	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 synchronization server and dbmlsync have different information about where the scanning should begin.
Publication_*n* (in)	publication name	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
Script version (in)	script version name	The MobiLink script version to be used for the synchronization.

Description

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

Actions of this procedure are committed immediately after execution.

See also

- "Synchronization event hook sequence" on page 177

Examples

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
 "event_id"            integer NOT NULL DEFAULT autoincrement ,
   "event_name"        varchar(128) NOT NULL ,
   "ml_user"             varchar(128) NULL ,
   "event_time"          timestamp NULL,
   "table_name"          varchar(128) NULL ,
   "upsert_count"        varchar(128) NULL ,
   "delete_count"        varchar(128) NULL ,
   "exit_code"           integer NULL ,
   "status_retval"       varchar(128) NULL ,
   "pubs"                  varchar(128) NULL ,
   "sync_descr "         varchar(128) NULL ,
    PRIMARY KEY ("event_id"),
)
```

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

```
  CREATE PROCEDURE sp_hook_dbmlsync_logscan_end ()
  BEGIN

   declare scan_retry varchar(128);

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

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

    IF scan_retry='true' then
     INSERT INTO SyncLog (event_name, ml_user,event_time,sync_
           descr)
      SELECT 'logscan_end', #hook_dict.value, current timestamp,
       'Transaction log rescanned'
      FROM #hook_dict
      WHERE name = 'MobiLink user' ;
     ELSE
     INSERT INTO SyncLog (event_name, ml_user,event_time,sync_
           descr)
      SELECT 'logscan_end', #hook_dict.value, current timestamp,
       'Transaction log scanned normally'
      FROM #hook_dict
      WHERE name = 'MobiLink user' ;
     END IF;
  END
```

sp_hook_dbmlsync_process_return_code

Function Use this stored procedure to manage return codes.

Rows in #hook_dict table

Name	Values	Description
Publication_*n* (in)	publication name	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
Fatal error (in)	**true** | **false**	True when the hook is called because of an error that will cause dbmlsync to terminate.
Aborted synchronization (in)	**true** | **false**	True when the hook is called because of an abort request from the sp_hook_dbmlsync_abort hook.
Return code (in)	number	The return 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 return code (in)	number	The value stored in the **new return 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 return code (in|out)	number	The desired return code for the process. When dbmlsync exits, dblmsync's **return code** is the value stored in this row by the last call to the hook. The value must be -32768 to 32767.

Name	Values	Description
Script version (in)	script version name	The MobiLink script version to be used for the synchronization.

Description

A dbmlsync session can run multiple synchronizations when you specify the -n option more than once on 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 return code does not indicate which failed. Use this hook to define the return code for the dbmlsync process based on the return codes from the synchronizations. This hook can also be used to log return codes.

Example

Suppose that you run dbmlsync to perform five synchronizations and you want the return code to indicate how many of the synchronizations failed, with a return code of 0 indicating that there were no failures, a return code of 1 indicating that one synchronization failed, and so on. You can achieve this by defining the sp_hook_dbmlsync_process_return_code hook as follows. In this case, if three synchronizations fail, the new return code is 3.

```
CREATE PROCEDURE sp_hook_dbmlsync_process_return_code()
BEGIN
   DECLARE   rc   integer;

   SELECT value INTO rc FROM #hook_dict WHERE name = 'return
         code';
   IF rc <> 0 THEN
      SELECT value INTO rc FROM #hook_dict WHERE name = 'last
         return code';
      UPDATE #hook_dict SET value = rc + 1 WHERE name = 'new
         return code';
      END IF;
END;
```

See also

- “Synchronization event hook sequence” on page 177
- “sp_hook_dbmlsync_abort” on page 183

sp_hook_dbmlsync_schema_upgrade

Function

Use this stored procedure to run a SQL script that revises your schema.

Rows in #hook_dict table

Name	Values	Description
Publication_*n* (in)	publication name	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
Script version	name of script version	The script version that you are synchronizing.
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.
Script version (in)	script version name	The MobiLink script version to be used for the synchronization.

Description

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 is required to ensure that the database will continue to be able to synchronize. When this hook is being used you should not set the dbmlsync extended option LockTables to off (LockTables is on by default).

During any synchronization where the upload was applied successfully and acknowledged by MobiLink, this hook is called after the sp_hook_dbmlsync_download_end hook and before the sp_hook_dbmlsync_end hook. This hook is not called during download-only synchronization or when a file-based download is being created or applied.

Actions performed in this hook are committed immediately after the hook completes.

See also

- "Schema changes in remote databases" [*MobiLink Administration Guide*, page 81]

Examples

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

```
CREATE PROCEDURE sp_hook_dbmlsync_schema_upgrade()
BEGIN
 -- Upgrade the schema of the Dealer table
 -- here we add a column
 ALTER TABLE Dealer
  ADD dealer_description varchar(128);

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

sp_hook_dbmlsync_set_extended_options

Function

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

Description

If a procedure of this name exists, it is called immediately before the sp_hook_dbmlsync_delay hook.

Extended options specified by this hook apply only to the synchronization identified by the publication and MobiLink user entries, and they apply only until the next time the hook is called for the same synchronization.

Scheduling options may not be specified using this hook.

Actions of this procedure are committed immediately after execution.

See also

- ♦ "Synchronization event hook sequence" on page 177
- ♦ "dbmlsync extended options" on page 105
- ♦ "Priority order for extended options and connection parameters" on page 73

Examples

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

```
CREATE PROCEDURE sp_hook_dbmlsync_set_extended_options ()
BEGIN
 IF exists(SELECT * FROM #hook_dict
  WHERE name LIKE ’publication_%’ AND value=’pub1’)
 THEN
   -- specify the sendcolumnnames=on extended option
   UPDATE #hook_dict
          SET value = ’SendColumnNames=on’
    WHERE name = ’extended options’;
 END IF;
END
```

sp_hook_dbmlsync_upload_begin

Function

Use this stored procedure to add custom actions immediately before the transmission of the upload.

Rows in #hook_dict table

Name	Values	Description
Publication_*n* (in)	publication name	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
Script version (in)	script version name	The MobiLink script version to be used for the synchronization.

Description

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

Actions of this procedure are committed immediately after execution.

See also

- "Synchronization event hook sequence" on page 177

Examples

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
 "event_id"            integer NOT NULL DEFAULT autoincrement ,
   "event_name"        varchar(128) NOT NULL ,
   "ml_user"             varchar(128) NULL ,
   "event_time"          timestamp NULL,
   "table_name"          varchar(128) NULL ,
   "upsert_count"        varchar(128) NULL ,
   "delete_count"        varchar(128) NULL ,
   "exit_code"           integer NULL ,
   "status_retval"       varchar(128) NULL ,
   "pubs"                  varchar(128) NULL ,
   "sync_descr "         varchar(128) NULL ,
    PRIMARY KEY ("event_id"),
)
```

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

```
CREATE PROCEDURE sp_hook_dbmlsync_upload_begin ()
BEGIN
 INSERT INTO SyncLog (event_name, ml_user,event_time)
  SELECT ’upload_begin’, #hook_dict.value, current timestamp
  FROM #hook_dict
  WHERE name = ’MobiLink user’;
END
```

sp_hook_dbmlsync_upload_end

Function

Use this stored procedure to add custom actions after dbmlsync has verified receipt of the upload stream by the MobiLink synchronization server.

Rows in #hook_dict table

Name	Values	Description
Failure cause (in)	See range of values in Description, below	The cause of failure of an upload. For more information, see Description.
Upload status (in)	**retry** | **committed** | **failed** | **unknown**	Specifies the status returned by the MobiLink synchronization server when dbmlsync attempted to verify receipt of the upload stream. **retry** The MobiLink synchronization server and dbmlsync had different values for the log offset from which the upload stream should start. The upload stream was not committed by the MobiLink synchronization server. The dbmlsync utility will attempt to send another upload stream starting from a new log offset. **committed** The upload stream was received by the MobiLink synchronization server and committed. **failed** The MobiLink synchronization server did not commit the upload stream. **unknown** The MobiLink synchronization server was started with the -tu option, causing transaction-level uploads. For each transaction that is uploaded, the sp_hook_dbmlsync_upload_begin and sp_hook_dbmlsync_upload_end hooks are called and the upload status value is **unknown** each time except the last one.

Name	Values	Description
Publication_*n* (in)	publication name	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
Script version (in)	script version name	The MobiLink script version to be used for the synchronization.
Authentication value (in)	value	This value is generated by the authenticate_user, authenticate_-user_hashed, or authenticate_-parameters script on the server. The value is an empty string when the upload status is unknown or when the upload_end hook is called after an upload is resent because of a conflict between the log offsets stored in the remote and consolidated databases.

Description

If a procedure of this name exists, it is called immediately after dbmlsync has sent the upload stream and received confirmation of it from the MobiLink synchronization server.

Actions of this procedure are committed immediately after execution.

The range of possible parameter values for the failure cause row in the #hook_dict table includes:

♦ **UPLD_ERR_COMMUNICATIONS_FAILURE** A communication error occurred.

♦ **UPLD_ERR_LOG_OFFSET_MISMATCH** The upload failed because of conflict between log offset stored on the remote and consolidated databases.

♦ **UPLD_ERR_GENERAL_FAILURE** The upload failed for an unknown reason.

♦ **UPLD_ERR_INVALID_USERID_OR_PASSWORD** The user ID or password was incorrect.

- **UPLD_ERR_USERID_OR_PASSWORD_EXPIRED** The user ID or password expired.
- **UPLD_ERR_USERID_ALREADY_IN_USE** The user ID was already in use.
- **UPLD_ERR_DOWNLOAD_NOT_AVAILABLE** The upload was committed on the consolidated but an error occurred that prevented MobiLink from generating a download stream.
- **UPLD_ERR_PROTOCOL_MISMATCH** *Dbmlsync* received unexpected data from the MobiLink synchronization server.
- **UPLD_ERR_SQLCODE_n** Here, *n* is an integer. A SQL error occurred in the consolidated database. The integer specified is the SQLCODE for the error encountered.

See also

- "Synchronization event hook sequence" on page 177

Examples

Assume you use the following table to log synchronization events on the remote database.

```
CREATE TABLE SyncLog
(
 "event_id"             integer NOT NULL DEFAULT autoincrement ,
   "event_name"         varchar(128) NOT NULL ,
   "ml_user"              varchar(128) NULL ,
   "event_time"           timestamp NULL,
   "table_name"           varchar(128) NULL ,
   "upsert_count"         varchar(128) NULL ,
   "delete_count"         varchar(128) NULL ,
   "exit_code"            integer NULL ,
   "status_retval"        varchar(128) NULL ,
   "pubs"                   varchar(128) NULL ,
   "sync_descr "          varchar(128) NULL ,
    PRIMARY KEY ("event_id"),
)
```

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

```
CREATE PROCEDURE sp_hook_dbmlsync_upload_end ()
BEGIN

 declare status_return_value varchar(255);

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

   INSERT INTO SyncLog (event_name, ml_user,
   status_retval, event_time)
  SELECT 'upload_end', #hook_dict.value,
   status_return_value, current timestamp
  FROM #hook_dict
  WHERE name = 'MobiLink user';
END
```

sp_hook_dbmlsync_validate_download_file

Function

Use this hook to implement custom logic to decide if a download file can be applied to the remote database.

Rows in #hook_dict table

Name	Values	Description
Publication_*n* (in)	publication name	The publications being synchronized, where *n* is an integer. There is one publication_*n* entry for each publication being uploaded. The *n* in publication_*n* and generation number_*n* match. The numbering of *n* starts at zero.
MobiLink user (in)	MobiLink user name	The MobiLink user for which you are synchronizing.
File last download time (in)		The download file's last download time. (The download file contains all rows that were changed between its last download time and its next last download time.)
File next last download time (in)		The download file's next last download time. (The download file contains all rows that were changed between its last download time and its next last download time.)
File creation time (in)		The time when the download file was created.
File generation number_*n* (in)	number	The generation numbers from the download file. There is one file generation number_*n* for each publication_*n* entry. The *n* in publication_*n* and generation number_*n* match. The numbering of *n* starts at zero.
User data (in)	string	The string specified with the dbmlsync -be option when the download file was created.

Name	Values	Description
Apply file (in|out)	**True**|**False**	If true (the default), the download file will be applied only if it passes dbmlsync's other validation checks. If false, the download file will not be 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 switch was used when the download file was created. If -bg was used, the generation numbers on the remote database are updated from the download file and normal generation number checks are not performed.

Description

Use this stored procedure to implement custom checks to decide if a download file can be applied.

If you want to compare the generation numbers or timestamps contained in the file with those stored in the remote database, they can be queried from the SYSSYNC and SYSPUBLICATION tables.

This hook is called when the -ba option is specified. It is called after the sp_hook_dbmlsync_upload_end hook and before the sp_hook_dbmlsync_download_begin hook.

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

See also

- "-be option" on page 101
- "-bg option" on page 102
- "File-Based Downloads" [*MobiLink Administration Guide,* page 85]

Examples

The following example prevents application of download files that don't contain the user string 'sales manager data'.

```
CREATE PROCEDURE sp_hook_dbmlsync_validate_download_file ()
BEGIN
  IF NOT exists(SELECT * FROM #hook_dict
   WHERE name = 'User data' AND value='sales manager data')
  THEN
    UPDATE #hook_dict
            SET value = 'false' WHERE name = 'Apply file';
  END IF;
END
```

CHAPTER 9

Dbmlsync Integration Component

About this chapter

This chapter describes how to use the Dbmlsync Integration Component. The Dbmlsync Integration Component helps you customize your Adaptive Server Anywhere client applications.

Contents

Introduction

The Dbmlsync Integration Component is useful for adding synchronization to your applications. It provides a set of properties, events, and methods to regulate the behavior of Adaptive Server Anywhere clients.

The Dbmlsync Integration Component is available in two forms, both of which expose the same properties, events and methods:

- ♦ A visual component that provides an easy way to integrate the standard dbmlsync user interface into your applications.
- ♦ A non-visual component that allows you to access the component's functionality with no user interface or with a custom user interface that you create yourself.

Using the Dbmlsync Integration Component, your application can initiate synchronization, receive information about the progress of a synchronization, and implement special processing based on synchronization events.

As an alternative to the Dbmlsync Integration Component, you can use the a_sync_db structure in the database tools library provided with Adaptive Server Anywhere.

☞ For more information about a_sync_db, see "a_sync_db structure" [*ASA Programming Guide*, page 298].

☞ For more information about Adaptive Server Anywhere clients, see "MobiLink synchronization client" on page 96.

Supported platforms

You can use the Dbmlsync Integration Component on all MobiLink supported Windows operating systems, including Windows CE versions supporting ActiveX.

Supported development environments include Microsoft Visual Basic 6.0, eMbedded Visual Basic, and Visual Studio .NET.

Setting up the Dbmlsync Integration Component

The Dbmlsync Integration Component can be used in a wide variety of programming environments. You should consult the documentation in your programming environment for information about how to set it up. Following are the procedures for setting up the integration component in Windows form applications and console applications.

Windows form applications

❖ To set up the Dbmlsync Integration Component in a Windows application

1. Add a reference to the component.
   - ♦ For the visual component, use *dbmlsynccomg.dll*, which is located in the *win32* subdirectory of your SQL Anywhere installation. Add it to the development environment toolbox.
   - ♦ For the non-visual component, use *dbmlsynccom.dll*, which is located in the *win32* subdirectory of your SQL Anywhere installation.
2. Create a component instance.
   - ♦ For the visual component, use the toolbox to drag and drop an instance on the form.
   - ♦ For the non-visual component, declare a global variable and create an instance in your initialization code.
     - For example, if you are using Visual Basic .NET, enter the following as a global variable declaration:

       ```
       Dim WithEvents dbmlsync1 As DbmlsyncCOM.Dbmlsync
       ```

       To create an instance, enter:

       ```
       dbmlsync1 = New DbmlsyncCOM.Dbmlsync
       ```
     - For example, if you are using C#, enter the following as a global variable declaration:

       ```
       private DbmlsyncCOM.Dbmlsync dbmlsync1;
       ```

       To create an instance, enter:

       ```
       dbmlsync1 = new DbmlsyncCOM.Dbmlsync();
       ```

       Note: for C# you must create the instance prior to assigning delegates for dbmlsync events (step 3 of this procedure).
3. Add functions to handle synchronization events.
   - ♦ For example, if you are using Visual Basic .NET, use the Handles keyword in your function prototype.

```
Private Sub dbmlsync1_ProgressMessage(ByVal msg As
         String)
Handles dbmlsync1.ProgressMessage
    lblProgressMessage.Text = msg
End Sub
```

♦ If you are using C#, write an event handler function.

```
private void dbmlsync1_ProgressMessage(string msg)
{
    lblProgressMessage.Text = msg;
}
```

In your C# initialization code, use delegates to specify the event handler function for a dbmlsync event. The following example assigns dbmlsync1_ProgressMessage, to handle the dbmlsync ProgressMessage event.

```
dbmlsync1.ProgressMessage +=
new DbmlsyncCOM._IDbmlsyncEvents_
         ProgressMessageEventHandler(dbmlsync1_
         ProgressMessage);
```

☞ For more information about events, see "Dbmlsync Integration Component events" on page 249.

4. Set properties.

   The following example sets the UploadEventsEnabled property to true. This enables the UploadRow event:

```
dbmlsync1.UploadEventsEnabled = True
```

☞ For more information about the UploadRow event, see "UploadRow event" on page 259.

☞ For more information about properties, see "Dbmlsync Integration Component properties" on page 244.

5. Use the Run method to begin one or more synchronizations. For example,

```
dbmlsync1.Run("-c eng=rem1;uid=dba;pwd=sql")
```

☞ For more information about the Run method, see "Run method" on page 243.

Console applications

Console applications can use the non-visual component only.

The fully qualified name of the Dbmlsync Integration Component is DbmlsyncCOM.Dbmlsync.

A console application that uses the non-visual Dbmlsync Integration Component must pump the Windows message queue. For example, with

.NET applications you must use the following methods in the System.Windows.Application namespace:

System.Windows.Application.Run() Starts a standard application message loop on the current thread.

System.Windows.Application.Exit() Requests all message pumps to terminate and then closes all application windows.

❖ To set up the Dbmlsync Integration Component in a console application (Visual Basic.NET)

1. Add a reference to *System.Windows.Forms.dll*.

2. Add a reference to the non-visual component, *dbmlsynccom.dll*, located in the *win32* subdirectory of your SQL Anywhere installation.

3. Create an instance of the Dbmlsync Integration Component.

   ♦ Enter the following as a global variable declaration:

   ```
   Dim WithEvents dbmlsync1 As DbmlsyncCOM.Dbmlsync
   ```

   ♦ To create a component instance, enter:

   ```
   dbmlsync1 = New DbmlsyncCOM.Dbmlsync
   ```

4. Set properties.

   The following example sets the UploadEventsEnabled property to true. This enables the UploadRow event:

   ```
   dbmlsync1.UploadEventsEnabled = True
   ```

   ☞ For more information about properties, see "Dbmlsync Integration Component properties" on page 244.

   ☞ For more information about the UploadRow event, see "UploadRow event" on page 259.

5. Use the Run method to begin one or more synchronizations. For example,

   ```
   dbmlsync1.Run("-c eng=rem1;uid=dba;pwd=sql")
   ```

   ☞ For more information about the Run method, see "Run method" on page 243.

6. Use System.Windows.Application Run method to enable pumping of the Windows message queue:

   ```
   System.Windows.Forms.Application.Run()
   ```

7. Add logic to handle synchronization events.

   ☞ For more information about events, see “Dbmlsync Integration Component events” on page 249.

8. Use the System.Windows.Application Exit method to request all message pumps to terminate:

   ```
   System.Windows.Application.Exit()
   ```

Dbmlsync Integration Component methods

The following are methods implemented by the DbmlsyncCOM.Dbmlsync class.

Run method

Function

Begins one or more synchronizations using dbmlsync command line options.

Syntax

Run(ByVal *cmdLine* As String **)**
Member of **DbmlsyncCOM.Dbmlsync**

Parameters

cmdLine A string specifying dbmlsync options.

Description

☞ For a list of options, see "dbmlsync options" on page 100.

The cmdLine parameter should contain the same options you would use if you were performing a synchronization with the dbmlsync command line utility. For example, running dbmlsync with the command line

```
dbmlsync -c uid=dba;pwd=sql
```

is equivalent to the following Run method invocation

```
dbmlsync1.Run "-c uid=dba;pwd=sql"
```

Example

The following example initiates a synchronization for a remote database called remote1.

```
dbmlsync1.Run "-c eng=remote1;uid=dba;pwd=sql"
```

Stop method

Function

Requests active synchronizations to terminate.

Syntax

Stop( )
Member of **DbmlsyncCOM.Dbmlsync**

Description

The Stop method issues a request to terminate any active synchronizations. It returns immediately.

The stop button built in to the visual Dbmlsync Integration Component automatically invokes this method.

Example

The following example stops synchronizations being run by the Dbmlsync Integration Component instance dbmlsync1.

```
dbmlsync1.Stop
```

Dbmlsync Integration Component properties

Dbmlsync Integration Component properties let you customize the behavior of the component and examine the state of a running synchronization.

Path property

Function	Specifies the location of *dbmlsync.exe*.
Syntax	Public Property **Path( )** As String Member of **DbmlsyncCOM.Dbmlsync**
Description	You do not need to set this property if *dbmlsync.exe* is located in a directory specified by the Windows PATH environment variable.
Example	The following example sets the path of a Dbmlsync Integration Component instance.

```
dbmlsync1.Path = "c:\program files\sybase\sql anywhere 9\win32"
```

UploadEventsEnabled property

Function	Enables the UploadRow event.
Syntax	Public Property **UploadEventsEnabled()** As Boolean Member of **DbmlsyncCOM.Dbmlsync**
Description	If you handle the UploadRow event, you should set this property to true. The default is false, which disables the UploadRow event. ☞ For more information about the UploadRow event, see "UploadRow event" on page 259.
Example	The following example sets UploadEventsEnabled to true:

```
  dbmlsync1.UploadEventsEnabled = True
```

DownloadEventsEnabled property

Function	Enables the DownloadRow event.
Syntax	Public Property **DownloadEventsEnabled( )** As Boolean Member of **DbmlsyncCOM.Dbmlsync**
Description	If you handle the DownloadRow event, you should set this property to true. The default is false, which disables the DownloadRow event. ☞ For more information about the DownloadRow event, see "DownloadRow event" on page 252.

Example

The following example sets DownloadEventsEnabled to true:

```
dbmlsync1.DownloadEventsEnabled = True
```

ErrorMessageEnabled property

Function

Prevents the Message event from being called for messages of type MsgError.

Syntax

Public Property **ErrorMessageEnabled( )** As Boolean
Member of **DbmlsyncCOM.Dbmlsync**

Description

If you do not handle error information in the Message event, you should set this property to false to improve performance. The default true, which enables messages of type MsgError to trigger the Message event.

☞ For more information about the Message event, see "Message event" on page 256.

Example

The following example sets ErrorMessageEnabled to false:

```
dbmlsync1.ErrorMessageEnabled = False
```

WarningMessageEnabled property

Function

Prevents the Message event from being called for messages of type MsgWarning.

Syntax

Public Property **WarningMessageEnabled( )** As Boolean
Member of **DbmlsyncCOM.Dbmlsync**

Description

If you do not handle warning information in the Message event, you should set this property to false to improve performance. The default is true, which enables messages of type MsgWarning to trigger the Message event.

☞ For more information about the Message event, see "Message event" on page 256.

Example

The following example sets WarningMessageEnabled to false:

```
dbmlsync1.WarningMessageEnabled = False
```

InfoMessageEnabled property

Function

Prevents the Message event from being called for messages of type MsgInfo.

Syntax

Public Property **InfoMessageEnabled( )** As Boolean
Member of **DbmlsyncCOM.Dbmlsync**

Description

If you do not handle general progress information in the Message event, you should set this property to false to improve performance. The default is true, which enables messages of type MsgInfo to trigger the Message event.

☞ For more information about the Message event, see "Message event" on page 256.

Example

The following example sets InfoMessageEnabled to false:

```
dbmlsync1.InfoMessageEnabled = False
```

DetailedInfoMessageEnabled property

Function

Prevents the Message event from being called for messages of type MsgDetailedInfo.

Syntax

Public Property **DetailedInfoMessageEnabled( )** As Boolean
Member of **DbmlsyncCOM.Dbmlsync**

Description

If you do not handle detailed progress information in the Message event, you should set this property to false to improve performance. The default is true, which enables messages of type MsgDetailedInfo to trigger the Message event.

☞ For more information about the Message event, see "Message event" on page 256.

Example

The following example sets DetailedInfoMessageEnabled to false:

```
dbmlsync1.DetailedInfoMessageEnabled = False
```

UseVB6Types property

Function

If you are using Visual Basic 6, set this property to true to simplify handling of row data returned by the UploadRow and DownloadRow events.

Syntax

Public Property **DetailedInfoMessageEnabled( )** As Boolean
Member of **DbmlsyncCOM.Dbmlsync**

Description

Visual Basic 6 does not support unsigned 32 bit values and any 64 bit values. Data of these types may be returned by the ColumnValue property of an IRowTransferData object. When UseVB6Types is set to true, data of these types is converted to other types supported by Visual Basic 6 for easier processing. Uint32 values are converted to double; 64 bit values are converted to strings.

☞ For more information about the IRowTransferData interface, see "IRowTransferData interface" on page 262.

☞ For more information about the UploadRow event, see "UploadRow event" on page 259.

☞ For more information about the DownloadRow event, see "DownloadRow event" on page 252.

Example — The following example enables data type coercion for a Dbmlsync Integration Component instance used in Visual Basic 6.0:

```
dbmlsync1.UseVB6Types = True
```

ExitCode property

Function — Returns the exit code from synchronizations started by the most recent Run method invocation.

Syntax — Public Property **ExitCode( )** As Integer
Member of **DbmlsyncCOM.Dbmlsync**

Description — The ExitCode property returns the exit code for the synchronizations started by the last Run method invocation. 0 indicates successful synchronizations. Any other value indicates that a synchronization failed.

> **Note:**
> Retrieving the value of this property before the DoneExecution event is triggered may result in a meaningless exit code value.

Example — The following example displays the exit code from the most recent synchronization attempt when the DoneExecution event is triggered.

```
Private Sub dbmlsync1_DoneExecution() Handles
         dbmlsync1.DoneExecution
        MsgBox(dbmlsync1.ExitCode)
    End Sub
```

EventChannelSize property

Function — Specifies the size of an internal buffer used for processing method calls.

Syntax — Public Property **EventChannelSize( )** As Integer
Member of **DbmlsyncCOM.Dbmlsync**

Description — Most users will never have to change this property.

DispatchChannelSize property

Function — Specifies the size of an internal buffer used for processing event information.

Syntax	Public Property **DispatchChannelSize( )** As Integer Member of **DbmlsyncCOM.Dbmlsync**
Description	Most users will never have to change this property.

Dbmlsync Integration Component events

Events provide a mechanism for client applications to receive and act on information about the progress of a synchronization.

BeginDownload event

The BeginDownload event is triggered at the beginning of the download stage of a synchronization.

Syntax

Public Event **BeginDownload( )**
Member of **DbmlsyncCOM.Dbmlsync**

Description

Use this event to add custom actions at the beginning of the download stage of a synchronization.

Example

The following Visual Basic .NET example outputs a message when the BeginDownload event is triggered.

```
Private Sub dbmlsync1_BeginDownload()
Handles dbmlsync1.BeginDownload

        MsgBox("Beginning Download")

End Sub
```

BeginLogScan event

The BeginLogScan event is triggered immediately before dbmlsync scans the transaction log to assemble the upload stream.

Syntax

Public Event **BeginLogScan(** ByVal *rescanLog* As Boolean **)**
Member of **DbmlsyncCOM.Dbmlsync**

Parameters

rescanLog If this is the first time the transaction log has been scanned for this synchronization, the value is false; otherwise it is true. The log is scanned twice when the MobiLink synchronization server and dbmlsync have different information about where scanning should begin.

Description

Use this event to add custom actions immediately before the transaction log is scanned for upload.

Example

The following Visual Basic .NET example outputs a message when the BeginLogScan event is triggered.

```
Private Sub dbmlsync1_BeginLogScan(
 ByVal rescanLog As Boolean
)
Handles dbmlsync1.BeginLogScan

        MsgBox("Begin Log Scan")

End Sub
```

BeginSynchronization event

The BeginSynchronization event is triggered at the beginning of each synchronization.

Syntax

Public Event **BeginSynchronization(_**
 ByVal *userName* As String, _
 ByVal *pubNames* As String _
)
Member of **DbmlsyncCOM.Dbmlsync**

Parameters

userName The MobiLink user for which you are synchronizing.

pubNames The publication being synchronized. If there is more than one publication this is a comma-separated list.

Description

Use this event to add custom actions at the beginning of a synchronization.

Example

The following Visual Basic .NET example outputs a message when the BeginSynchronization event is triggered. The message outputs the user and publication names.

```
Private Sub dbmlsync1_BeginSynchronization(
 ByVal userName As String,
 ByVal pubNames As String
)
Handles dbmlsync1.BeginSynchronization

       MsgBox("Beginning synchronization for: " + userName _
     + " publication: " + pubNames)

End Sub
```

BeginUpload event

The BeginUpload event is triggered immediately before the transmission of the upload.

Syntax

Public Event **BeginUpload( )**
Member of **DbmlsyncCOM.Dbmlsync**

Description

Use this event to add custom actions immediately before the transmission of the upload to the MobiLink synchronization server.

Example

The following Visual Basic .NET example outputs a message when the BeginUpload event is triggered.

```
Private Sub dbmlsync1_BeginUpload()
Handles dbmlsync1.BeginUpload

        MsgBox("Begin Upload")

End Sub
```

ConnectMobilink event

The ConnectMobilink event is triggered immediately before the component connects to the MobiLink synchronization server.

Syntax

Public Event **ConnectMobilink( )**
Member of **DbmlsyncCOM.Dbmlsync**

Description

Use this event to add custom actions immediately before the remote database connects to the MobiLink synchronization server. At this stage, dbmlsync has generated the upload stream.

The ConnectMobiLink event occurs after the BeginSynchronization event.

Example

The following Visual Basic .NET example outputs a message when the ConnectMobilink event is triggered.

```
Private Sub dbmlsync1_ConnectMobilink()
Handles dbmlsync1.ConnectMobilink

        MsgBox("Connecting to the MobiLink synchronization
         server")

End Sub
```

DisconnectMobilink event

The DisconnectMobilink event is triggered immediately after the component disconnects from the MobiLink synchronization server.

Syntax

Public Event **DisconnectMobilink( )**
Member of **DbmlsyncCOM.Dbmlsync**

Description

Use this event to add custom actions immediately after the remote database disconnects from the MobiLink synchronization server.

Example

The following Visual Basic .NET example outputs a message when the DisconnectMobilink event is triggered.

```
Private Sub dbmlsync1_DisconnectMobilink()
Handles dbmlsync1.DisconnectMobilink

        MsgBox("Disconnected from the MobiLink synchronization
         server")

End Sub
```

DoneExecution event

The DoneExecution event is triggered when all synchronizations started by a Run method invocation have completed.

Syntax

Public Event **DoneExecution( )**
Member of **DbmlsyncCOM.Dbmlsync**

Description

Use this event to add custom actions when all synchronizations started by a Run method invocation have completed.

Example

Using the ExitCode property, the following Visual Basic .NET example outputs the exit code from the synchronizations started by the last Run method invocation:

```
Private Sub dbmlsync1_DoneExecution()
Handles dbmlsync1.DoneExecution

        MsgBox(dbmlsync1.ExitCode)

End Sub
```

DownloadRow event

The DownloadRow event is triggered when a row is downloaded from the MobiLink synchronization server.

Syntax

Public Event **DownloadRow(**
  ByVal *rowData* As DbmlsyncCOM.IRowTransferData
)
Member of **DbmlsyncCOM.Dbmlsync**

Parameters

rowData An IRowTransferData object containing detailed information about the downloaded row.

☞ For more information about the IRowTransferData interface, see "IRowTransferData interface" on page 262.

Description

Use this event to examine rows being downloaded from the MobiLink synchronization server.

To enable the DownloadRow event, use the DownloadEventsEnabled property.

☞ For more information, see “DownloadEventsEnabled property” on page 244.

When a delete operation is encountered in the download row event, only primary key column values are available.

Example

The following Visual Basic .NET example iterates through all the columns for a row in the DownloadRow event. It determines if a column value is null, and outputs column names and values.

```
Private Sub dbmlsync1_DownloadRow(
 ByVal rowData As DbmlsyncCOM.IRowTransferData
)
Handles dbmlsync1.DownloadRow

Dim liX As Integer
For liX = 0 To rowData.ColumnCount - 1
    If VarType(rowData.ColumnValue(liX)) <> VariantType.Null
         Then
        ’ output the non-null column value
        MsgBox("Column " + CStr(liX) + ": " +
         rowData.ColumnName(liX) + _
        ", " + CStr(rowData.ColumnValue(liX)))
    Else
        ’ output ’NULL’ for the column value
        MsgBox("Column " + CStr(liX) + ": " +
         rowData.ColumnName(liX) + _
        ", " + "NULL")
    End If
Next liX

End Sub
```

EndDownload event

The EndDownload event is triggered at the end of the download stage of the synchronization process.

Syntax

Public Event **EndDownload(**
 long *upsertRows*,
 long *deleteRows*
)
Member of **DbmlsyncCOM.Dbmlsync**

Parameters

upsertRows Indicates the number of rows updated or inserted by the download.

deleteRows Indicates the number of rows deleted by the download.

Description

Use this event to add custom actions at the end of the download stage of synchronization.

Example

The following Visual Basic .NET example outputs a message and the number of inserted, updated, and deleted rows when the EndDownload event is triggered.

```
Private Sub dbmlsync1_EndDownload(
 ByVal upsertRows As Integer,
 ByVal deleteRows As Integer
)
Handles dbmlsync1.EndDownload

    MsgBox("Download complete." + _
     CStr(upsertRows) + "Rows updated or inserted" + _
     CStr(deleteRows) + "Rows deleted")

End Sub
```

EndLogScan event

The EndLogScan event is triggered immediately after the transaction log is scanned for upload.

Syntax

Public Event **EndLogScan( )**
Member of **DbmlsyncCOM.Dbmlsync**

Description

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

Example

The following Visual Basic .NET example outputs a message when the EndLogScan event is triggered.

```
Private Sub dbmlsync1_EndLogScan()
Handles dbmlsync1.EndLogScan

        MsgBox("Scan of transaction log complete...")

End Sub
```

EndSynchronization event

The EndSynchronization event is triggered when a synchronization is complete.

Syntax

Public Event **EndSynchronization(**
 ByVal *exitCode* As Integer,
 ByRef *restart* As Boolean
)
Member of **DbmlsyncCOM.Dbmlsync**

Parameters

exitCode If set to anything other than zero, this indicates that a synchronization error occurred.

restart This value is set to false when the event is called. If the event changes its value to true, dbmlsync will restart the synchronization.

Description

Use this event to add custom actions when a synchronization is complete.

Example

The following Visual Basic .NET example uses the EndSynchronization event to restart up to five failed synchronization attempts. If all restart attempts failed, the message “All restart attempts failed” is output, along with the exit code. If a synchronization is successful, the message “Synchronization succeeded “ is output, along with the exit code.

```
’ Global variable for the number of restarts
Dim numberOfRestarts As Integer

Private Sub dbmlsync1_EndSynchronization(
 ByVal ExitCode As Integer,
 ByRef restart As Boolean
)
Handles dbmlsync1.EndSynchronization

    If numberOfRestarts < 5 Then
        MsgBox("Restart Number: " + CStr(numberOfRestarts + 1))
        If ExitCode <> 0 Then
            ’ restart the failed synchronization
            restart = True
            numberOfRestarts = numberOfRestarts + 1
        Else
            ’ the last synchronization succeeded
            MsgBox("Synchronization succeeded. " + _
              "Exit code: " + CStr(ExitCode))
        End If
    Else
        MsgBox("All restart attempts failed. " + _
             "Exit code: " + CStr(ExitCode))
    End If

End Sub
```

EndUpload event

The EndUpload event is triggered immediately after transmission of the upload stream to the MobiLink synchronization server.

Syntax

Public Event **EndUpload( )**
Member of **DbmlsyncCOM.Dbmlsync**

Description

Use this event to add custom actions immediately after transmission of the upload stream from dbmlsync to the MobiLink synchronization server.

Example

The following Visual Basic .NET example outputs a message when the

EndUpload event is triggered.

```
Private Sub dbmlsync1_EndUpload()
Handles dbmlsync1.EndUpload

    MsgBox("End Upload")

End Sub
```

Message event

The Message event is triggered when dbmlsync logs information.

Syntax

Public Event **Message(_**
 ByVal *msgClass* As DbmlsyncCOM.MessageClass, _
 ByVal *msgID* As Integer, ByVal *msg* As String_
)
Member of **DbmlsyncCOM.Dbmlsync**

Parameters

msgClass indicates the severity of the message. Values can be:

- **MsgInfo** A message containing progress information about the synchronization.
- **MsgDetailedInfo** Like MsgInfo, but containing more detailed information.
- **MsgWarning** A message indicating a potential problem but one that will not prevent successful synchronization.
- **MsgError** A message indicating a problem that will prevent successful synchronization.

msgID A unique identifier for the message. If msgID is zero, the message does not have a unique identifier.

msg The text of the message.

Description

Use this event to receive information logged by dbmlsync.

Example

The following Visual Basic .NET example adds messages logged by dbmlsync to a listbox control.

```
Private Sub dbmlsync1_Message(
 ByVal msgClass As DbmlsyncCOM.MessageClass,
 ByVal msgId As Integer, ByVal msg As String
)
Handles dbmlsync1.Message

    Select Case msgClass
        Case DbmlsyncCOM.MessageClass.MsgError
            lstMessages.Items.Add("Error: " + msg)
        Case DbmlsyncCOM.MessageClass.MsgWarning
            lstMessages.Items.Add("Warning: " + msg)
        Case DbmlsyncCOM.MessageClass.MsgInfo
            lstMessages.Items.Add("Info: " + msg)
    End Select

End Sub
```

ProgressIndex event

The ProgressIndex event is triggered when dbmlsync updates its progress bar.

Syntax

Public Event **ProgressIndex(_**
 ByVal *index* As Integer, _
 ByVal *max* As Integer _
)
Member of **DbmlsyncCOM.Dbmlsync**

Parameters

index An integer representing the progress of the synchronization.

max The maximum progress value. If this value is zero, the maximum value has not changed since the last time the event was fired.

Description

Use this event to update a progress indicator such as a progress bar.

Example

The following Visual Basic .NET example updates a progress bar control based on the Index value. The maximum index value is set at the beginning of the synchronization.

```
Private Sub dbmlsync1_ProgressIndex(
 ByVal index As Integer,
 ByVal max As Integer
)
Handles dbmlsync1.ProgressIndex

    If max <> 0 Then
        ProgressBar1.Maximum = max
    End If
    ProgressBar1.Value = index

End Sub
```

ProgressMessage event

The ProgressMessage event is triggered when synchronization progress information changes.

Syntax

Public Event **ProgressMessage(** ByVal *msg* As String **)**
Member of **DbmlsyncCOM.Dbmlsync**

Parameters

msg The new progress string.

Description

Use this event to receive the string normally displayed above the dbmlsync progress bar.

Example

The following Visual Basic .NET example sets the value of a progress label when the ProgressMessage event is triggered.

```
Private Sub dbmlsync1_ProgressMessage(
 ByVal msg As String
)
Handles dbmlsync1.ProgressMessage

    lblProgressMessage.Text = msg

End Sub
```

SetTitle event

The SetTitle event is triggered when status information changes. In the dbmlsync utility, this information is displayed in the title bar.

Syntax

Public Event **SetTitle(** ByVal *title* **)** As String
)
Member of **DbmlsyncCOM.Dbmlsync**

Parameters

title The title in the dbmlsync window title bar.

Description

Use this event to receive the title normally seen on the dbmlsync window when its value changes.

Example

The following Visual Basic .NET example sets the title of a Windows form when the SetTitle event is triggered.

```
Private Sub dbmlsync1_SetTitle(
 ByVal title As String
)
Handles dbmlsync1.SetTitle

        Me.Text = title

End Sub
```

UploadAck event

The UploadAck event is triggered after the component has received acknowledgement of the upload stream from the MobiLink synchronization server.

Syntax

Public Event **UploadAck(** _
  ByVal *status* As DbmlsyncCOM.UploadAckStatus _
)
Member of **DbmlsyncCOM.Dbmlsync**

Parameters

status Indicates the status returned by MobiLink to the remote after the upload stream is processed. Its value is one of:

- **StatCommitted** Indicates that the upload stream was received by the MobiLink synchronization server and committed.
- **StatRetry** Indicates that the MobiLink synchronization server and the remote database had different values for the log offset from which the upload stream should start. The upload stream was not committed by the MobiLink synchronization server. The component will attempt to send another upload stream starting from the MobiLink synchronization server's log offset.
- **StatFailed** Indicates that the MobiLink synchronization server did not commit the upload stream.

Description

Use this event to add custom actions after dbmlsync has received acknowledgement of the upload stream from the MobiLink synchronization server.

Example

The following Visual Basic .NET example outputs a message if the upload has failed when the UploadAck event is triggered.

```
Private Sub dbmlsync1_UploadAck(ByVal status As
         DbmlsyncCOM.UploadAckStatus) Handles
         dbmlsync1.UploadAck

    If status = DbmlsyncCOM.UploadAckStatus.StatFailed Then
         MsgBox("Upload Failed")
    End If

End Sub
```

UploadRow event

The UploadRow event is triggered when a row is uploaded to the MobiLink synchronization server.

Syntax

Public Event **UploadRow(**
 ByVal *rowData* As DbmlsyncCOM.IRowTransferData
)
Member of **DbmlsyncCOM.Dbmlsync**

Parameters

rowData An IRowTransferData object containing detailed information about the uploaded row.

☞ For more information about the IRowTransferData interface, see "IRowTransferData interface" on page 262.

Description

Use this event to examine rows being uploaded to the MobiLink synchronization server.

To enable the UploadRow event, use the UploadEventsEnabled property.

☞ For more information, see "UploadEventsEnabled property" on page 244.

Example

The following Visual Basic .NET example iterates through all the columns for a row in the UploadRow event. It determines if a column value is null and outputs column names and values.

```
Private Sub dbmlsync1_UploadRow(
 ByVal rowData As DbmlsyncCOM.IRowTransferData
)
Handles dbmlsync1.UploadRow

Dim liX As Integer
For liX = 0 To rowData.ColumnCount - 1
    If VarType(rowData.ColumnValue(liX)) <> VariantType.Null
         Then
        ' output the non-null column value
        MsgBox("Column " + CStr(liX) + ": " +
         rowData.ColumnName(liX) + _
        ", " + CStr(rowData.ColumnValue(liX)))
    Else
        ' output 'NULL' for the column value
        MsgBox("Column " + CStr(liX) + ": " +
         rowData.ColumnName(liX) + _
        ", " + "NULL")
    End If
Next liX

End Sub
```

WaitingForUploadAck event

The WaitingForUploadAck event is triggered when the component begins waiting for upload acknowledgement from the MobiLink synchronization server.

Syntax

Public Event **WaitingForUploadAck( )**
Member of **DbmlsyncCOM.Dbmlsync**

Description

Use this event to add custom actions when the component is waiting for upload acknowledgement from the MobiLink synchronization server.

Example

The following Visual Basic .NET example outputs a message when the WaitingForUploadAck event is triggered.

```
Private Sub dbmlsync1_WaitingForUploadAck()
Handles dbmlsync1.WaitingForUploadAck

    MsgBox("Waiting for Upload Acknowledgement")

End Sub
```

IRowTransferData interface

Public Interface **IRowTransferData**
Member of **DbmlsyncCOM**

The UploadRow and DownloadRow events accept DbmlsyncCOM.IRowTransferData objects as parameters to examine uploaded and downloaded rows. This interface defines detailed row information including the table name, row operation, and column names.

RowOperation property

Function

Specifies the operation performed on the row.

Syntax

Public Property **RowOperation( )** As DbmlsyncCOM.RowEventOp
Member of **DbmlsyncCOM.IRowTransferData**

Description

This property has one of the following values:

OpInsert The row was inserted.

OpUpdate The row was updated.

OpDelete The row was deleted.

OpTruncate The table was truncated (all the rows in the table were deleted). When the RowOperation property has this value, the ColumnName and ColumnValue properties return invalid information.

Note: For the DownloadRow event, upsert (update or insert) operations are given the OpInsert value.

TableName property

Function

The name of the table on which an upload or download operation occurred.

Syntax

Public Property **TableName( )** As String
Member of **DbmlsyncCOM.IRowTransferData**

Description

The TableName property specifies the name of the table on which an upload or download operation occurred. The following example illustrates the use of the TableName property in the UploadRow event.

☞ For more information about the UploadRow event, see "UploadRow event" on page 259.

Example

Following is a Visual Basic .NET example.

```
Private Sub dbmlsync1_UploadRow(
 ByVal rowData As DbmlsyncCOM.IRowTransferData
)
Handles dbmlsync1.UploadRow

    MsgBox ("Table name:" + rowData.TableName)

End Sub
```

ColumnName property

Function

Retrieves the column names for a row on which an upload or download operation occurred.

Syntax

Public Property **ColumnName(**ByVal *index* As String**)** As Object
Member of **DbmlsyncCOM.IRowTransferData**

Parameters

index   A zero based integer specifying the column name to be retrieved. Index values range from zero to one less than the ColumnCount property value.

☞ For more information about the ColumnCount property, see "ColumnCount property" on page 265.

Description

Associated column values can be retrieved using the ColumnValue property with the same index.

Example

The following Visual Basic .NET example iterates through all the columns for a row in the UploadRow event. It determines if a column value is null and outputs column names and values.

☞ For more information about the UploadRow event, see "UploadRow event" on page 259.

```
Private Sub dbmlsync1_UploadRow(
 ByVal rowData As DbmlsyncCOM.IRowTransferData
)
Handles dbmlsync1.UploadRow

Dim liX As Integer
For liX = 0 To rowData.ColumnCount - 1
    If VarType(rowData.ColumnValue(liX)) <> VariantType.Null
         Then
        ’ output the non-null column value
        MsgBox("Column " + CStr(liX) + ": " +
         rowData.ColumnName(liX) + _
        ", " + CStr(rowData.ColumnValue(liX)))
    Else
        ’ output ’NULL’ for the column value
        MsgBox("Column " + CStr(liX) + ": " +
         rowData.ColumnName(liX) + _
        ", " + "NULL")
    End If
Next liX

End Sub
```

ColumnValue property

Function

Retrieves the value of columns on which an upload or download operation occurred.

Syntax

Public Property **ColumnValue(** ByVal *index* As Integer **)** As Object
Member of **DbmlsyncCOM.IRowTransferData**

Parameters

index The zero based integer specifying the column value to be retrieved. Index values range from zero to one less than the ColumnCount property value.

☞ For more information about the ColumnCount property, see “ColumnCount property” on page 265.

Description

When an update operation is encountered, the column values given by this property are the values after the update is applied.

Associated column names can be retrieved using the ColumnName property with the same index.

Blob column values are not available through this property. When a blob column is encountered, the ColumnValue is the string “(blob)”.

Example

The following Visual Basic .NET example iterates through all the columns for a row in the UploadRow event. It determines if a column value is null and outputs column names and values.

☞ For more information about the UploadRow event, see "UploadRow event" on page 259.

```
Private Sub dbmlsync1_UploadRow(
 ByVal rowData As DbmlsyncCOM.IRowTransferData
)
Handles dbmlsync1.UploadRow

Dim liX As Integer
For liX = 0 To rowData.ColumnCount - 1
    If VarType(rowData.ColumnValue(liX)) <> VariantType.Null
         Then
        ' output the non-null column value
        MsgBox("Column " + CStr(liX) + ": " +
         rowData.ColumnName(liX) + _
        ", " + CStr(rowData.ColumnValue(liX)))
    Else
        ' output 'NULL' for the column value
        MsgBox("Column " + CStr(liX) + ": " +
         rowData.ColumnName(liX) + _
        ", " + "NULL")
    End If
Next liX

End Sub
```

ColumnCount property

Function

The number of columns contained in a row on which an upload or download operation occurred.

Syntax

Public Property **ColumnCount( )** As Integer
Member of **DbmlsyncCOM.IRowTransferData**

Description

The ColumnCount property specifies the number of columns for a row on which an upload or download operation occurred. The following example illustrates the use of the ColumnCount property in the UploadRow event.

☞ For more information about the UploadRow event, see "UploadRow event" on page 259.

Example

Following is a Visual Basic .NET example.

```
Private Sub dbmlsync1_UploadRow(
 ByVal rowData As DbmlsyncCOM.IRowTransferData
)
Handles dbmlsync1.UploadRow

    MsgBox "Number of Columns:" + CStr(rowData.ColumnCount)

End Sub
```

CHAPTER 10

DBTools Interface for dbmlsync

About this chapter

This chapter describes how to configure and use the DBTools interface for dbmlsync, allowing you to customize your Adaptive Server Anywhere client applications.

Contents

Introduction

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.

☞ For more information about the DBTools library, see "The Database Tools Interface" [*ASA Programming Guide,* page 259].

You can use the DBTools interface for dbmlsync to integrate synchronization functionality into your MobiLink synchronization client applications. For example, you can use the interface to display dbmlsync output messages in a custom user interface.

As an alternative to the DBTools interface for dbmlsync, you can use the Dbmlsync Integration Component.

☞ For more information about the Dbmlsync Integration Component, see "About this chapter" on page 237.

The DBTools interface for dbmlsync consists of the following elements that let you configure and run the MobiLink synchronization client:

- ♦ **a_sync_db structure** This structure holds settings, corresponding to dbmlsync command line options, that allow you to customize synchronization. This structure also contains pointers to callback functions receiving synchronization and progress information.

  ☞ For more information about a_sync_db, see "a_sync_db structure" [*ASA Programming Guide,* page 298].

- ♦ **a_syncpub structure** This structure holds publication information. You can specify a linked list of publications for synchronization.

  ☞ For more information about a_syncpub, see "a_syncpub structure" [*ASA Programming Guide,* page 306].

- ♦ **DBSynchronizeLog function** This function starts the synchronization process. Its only parameter is a pointer to an a_sync_db instance.

  ☞ For more information about DBSynchronizeLog, see "DBSynchronizeLog function" [*ASA Programming Guide,* page 275].

Setting up the DBTools interface for dbmlsync

This section guides you through the basic steps for using the DBTools interface for dbmlsync.

☞ For more information about the DBTools library, see "Introduction to the database tools interface" [*ASA Programming Guide,* page 260].

☞ For more information about using import libraries for your development environment, see "Using the database tools interface" [*ASA Programming Guide,* page 261].

❖ To configure and start dbmlsync using the DBTools interface

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 start DBTools.

   ```
   ret = DBToolsInit( &info );
   if( ret != 0 ) {
    printf("dbtools initialization failure \n");
   }
   ```

   ☞ For more information about DBTools initialization, see:

   - "Using the database tools interface" [*ASA Programming Guide,* page 261].
   - "a_dbtools_info structure" [*ASA Programming Guide,* page 294].
   - "DBToolsInit function" [*ASA Programming Guide,* page 275].

3. Initialize the a_sync_db instance.

   ♦ Declare an a_sync_db instance. For example, declare an instance called dbsync_info:

```
a_sync_db dbsync_info;
```

- Clear a_sync_db structure fields.

  ```
  memset( &dbsync_info, 0, sizeof( dbsync_info ) );
  ```

- Set required a_sync_db fields.

  ```
  dbsync_info.version = DB_TOOLS_VERSION_NUMBER;
  dbsync_info.output_to_mobile_link = 1;
  dbsync_info.default_window_title
    = "dbmlsync dbtools sample";
  ```

- Set the database connection string.

  ```
  dbsync_info.connectparms = "uid=dba;pwd=sql";
  ```

  ☞ For more information about database connection parameters, see "-c option" on page 102.

- Set other a_sync_db fields to customize synchronization.

  Most fields correspond to dbmlsync command line options. For more information about this correspondence, see *dbtools.h*.

  In the example below, verbose operation is enabled.

  ```
  dbsync_info.verbose_upload = 1;
  dbsync_info.verbose_option_info = 1;
  dbsync_info.verbose_row_data = 1;
  dbsync_info.verbose_row_cnts = 1;
  ```

- Initialize other a_sync_db fields.

☞ For more information about a_sync_db fields, see "a_sync_db structure" [*ASA Programming Guide,* page 298].

4. Create callback functions to receive feedback during synchronization and assign these functions to the appropriate a_sync_db fields.

   The following functions use the standard output stream to display dbmlsync error, log, and progress information.

   ☞ For more information about DBTools callback functions, see "Using callback functions" [*ASA Programming Guide,* page 263].

   - For example, create a function called dbsyncErrorCallBack to handle generated error messages:

   ```
   extern short _callback dbsyncErrorCallBack( char *str )
   {
       if( str != NULL ) {
           printf( "Error Msg    %s\n", str );
       }
       return 0;
   }
   ```

♦ For example, create a function called dbsyncWarningCallBack to handle generated warning messages:

```
extern short _callback dbsyncWarningCallBack( char *str
         )
{
    if( str != NULL ) {
        printf( "Warning Msg  %s\n", str );
    }
    return 0;
}
```

♦ For example, create a function called dbsyncLogCallBack to receive verbose informational messages that you might choose to log to a file instead of displaying in a window:

```
extern short _callback dbsyncLogCallBack( char *str )
{
    if( str != NULL ) {
        printf( "Log Msg      %s\n", str );
    }
    return 0;
}
```

♦ For example, create a function called dbsyncMsgCallBack to receive informational messages generated during synchronization.

```
extern short _callback dbsyncMsgCallBack( char *str )
{
    if( str != NULL ) {
        printf( "Display Msg  %s\n", str );
    }
    return 0;
}
```

♦ For example, create a function called dbsyncProgressMessageCallBack to receive the progress text. In the dbmlsync utility, this text is displayed directly above the progress bar.

```
extern short _callback dbsyncProgressMessageCallBack(
 char *str )
{
    if( str != NULL ) {
        printf( "ProgressText %s\n", str );
    }
    return 0;
}
```

♦ For example, create a function called dbsyncProgressIndexCallBack to receive information for updating a progress indicator or progress bar. This function receives two parameters:

- **index** An integer representing the current progress of a synchronization.

* **max** The maximum progress value. If this value is zero, the maximum value has not changed since the last time the event was fired.

```
extern short _callback dbsyncProgressIndexCallBack
(a_sql_uint32 index, a_sql_uint32 max )
{
    printf( "ProgressIndex    Index %d Max: %d\n",
           index, max );
    return 0;
}
```

A typical sequence of calls to this callback is shown below

```
// example calling sequence
// dbsyncProgressIndexCallBack( 0, 100 );
//   dbsyncProgressIndexCallBack( 25, 0 );
//   dbsyncProgressIndexCallBack( 50, 0 );
//   dbsyncProgressIndexCallBack( 75, 0 );
//   dbsyncProgressIndexCallBack( 100, 0 );
```

This sequence should result in the progress bar being set to 0% done, 25% done, 50% done, 75% done, and 100% done.

♦ For example, create a function called dbsyncWindowTitleCallBack to receive status information. In the dbmlsync utility, this information is displayed in the title bar.

```
extern short _callback dbsyncWindowTitleCallBack(
 char *title )
{
    printf( "Window Title     %s\n", title );
    return 0;
}
```

♦ The dbsyncMsgQueueCallBack function is called when a delay or sleep is required. It must return one of the following values, which are defined in *dllapi.h*.

* **MSGQ_SLEEP_THROUGH** indicates that the routine slept for the requested number of milliseconds. In most cases this is the value you should return.
* **MSGQ_SHUTDOWN_REQUESTED** indicates that you would like the synchronization to terminate as soon as possible.
* **MSGQ_SYNC_REQUESTED** indicates that the routine slept for less than the requested number of milliseconds and that the next synchronization should begin immediately if a synchronization is not currently in progress.

```
extern short _callback dbsyncMsgQueueCallBack(
         a_sql_uint32 sleep_period_in_milliseconds )
{

 printf( "Sleep %d ms\n", sleep_period_in_milliseconds
         );
 Sleep( sleep_period_in_milliseconds );
 return MSGQ_SLEEP_THROUGH;
 }
```

- Assign callback function pointers to the appropriate a_sync_db synchronization structure fields.

```
// set call back functions
dbsync_info.errorrtn    = dbsyncErrorCallBack;
dbsync_info.warningrtn  = dbsyncWarningCallBack;
dbsync_info.logrtn      = dbsyncLogCallBack;
dbsync_info.msgrtn      = dbsyncMsgCallBack;
dbsync_info.msgqueuertn = dbsyncMsgQueueCallBack;
dbsync_info.progress_index_rtn
     = dbsyncProgressIndexCallBack;
dbsync_info.progress_msg_rtn
     = dbsyncProgressMessageCallBack;
dbsync_info.set_window_title_rtn
     = dbsyncWindowTitleCallBack;
```

5. Create a linked list of a_syncpub structures to specify which publications should be synchronized.

   Each node in the linked list corresponds to one instance of the -n option on the dbmlsync command line.

   - Declare an a_syncpub instance. For example, call it publication_info:

   ```
   a_syncpub publication_info;
   ```

   - Initialize a_syncpub fields, specifying publications you want to synchronize.

     For example, to identify the template_p1 and template_p2 publications together in a single synchronization session:

   ```
   publication_info.next = NULL; // linked list terminates
   publication_info.pub_name = "template_p1,template_p2";
   publication_info.ext_opt  = "sv=template_ver1";
   publication_info.alloced_by_dbsync = 0;
   ```

     This is equivalent to specifying `-n template_p1,template_p2` on the dbmlsync command line.

     The associated script version specified using the ext_opt field, provides the same functionality as the dbmlsync -eu option.

     ☞ For more information, see "-eu option" on page 140.

♦ 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 corresponds to using the dbmlsync -n option.

☞ For more information about the dbmlsync -n option, see "-n option" on page 142.

☞ For more information about the a_syncpub structure, see "a_syncpub structure" [*ASA Programming Guide,* page 306].

6. Run dbmlsync using the DBSynchronizeLog function.

   In the following code listing, sync_ret_val is a return value as listed in "Software component return codes" [*ASA Programming Guide,* page 263].

```
printf("Running dbmlsync using dbtools interface...\n");
sync_ret_val = DBSynchronizeLog(&dbsync_info);
 printf("\n Done... synchronization return value is: %I \n",
         sync_ret_val);
```

   You can repeat step 6 multiple times with the same or different parameter values.

7. Shutdown the DBTools interface.

   The DBToolsFini function frees DBTools resources.

```
  DBToolsFini( &info );
```

   ☞ For more information about the DBToolsFini function, see "DBToolsFini function" [*ASA Programming Guide,* page 275].

PART III

ULTRALITE CLIENTS

This part contains material that describes how to set up and run UltraLite clients for MobiLink synchronization.

CHAPTER 11

UltraLite Clients

About this chapter

This chapter describes how to use an UltraLite database as a MobiLink client. It provides information on designing UltraLite databases to make the most of synchronization. It also introduces network protocols and provides material on how to set up Palm OS devices and Windows CE devices for synchronization.

☞ For more information about UltraLite, see *UltraLite Database User's Guide*.

☞ For information about how to use Adaptive Server Anywhere databases as MobiLink clients, see "Adaptive Server Anywhere Clients" on page 59.

Contents

Introduction

UltraLite developers can synchronize the data in UltraLite databases with a central consolidated database. This database may be a desktop database for personal applications, or a multi-user database for shared enterprise data.

Adding synchronization to your UltraLite application

Mobile and embedded databases generally cannot contain all the data that exists in the consolidated database. In practice, however, the only data you need locally is that used by the particular application you wish to make mobile. UltraLite provides the ability to take such a piece of a database, and keep it synchronized with the consolidated database.

The tables in each UltraLite database can have a subset of the rows and columns in the central database. For example, a customer table might contain over 100 columns and 100 000 rows in the consolidated database, but the UltraLite database may only require 4 columns and 1000 rows. MobiLink allows you to define the exact subset to be downloaded to each remote database.

❖ To add synchronization to an UltraLite application

1. Prepare the synchronization stream.

   Select a network protocol and set protocol options as required for that protocol.

   ☞ For more information, see "Selecting a network protocol" on page 288.

2. Call the synchronization function.

   The synchronization function depends on the development model you are using and the network protocol the application is using.

   ☞ For more information, see "Calling the synchronization function" on page 289.

Separately licensable option required
Transport-layer security requires that you obtain the separately-licensable SQL Anywhere Studio security option and is subject to export regulations.

☞ To order this component, see "Separately-licensable components" [*Introducing SQL Anywhere Studio*, page 5].

Choosing data to synchronize

By default, when you add synchronization to your database, all data is synchronized. You can customize what data is synchronized using publications, as well as a variety of other techniques.

Designing sets of data to synchronize separately

The schema of an UltraLite database is defined by the queries included in the application. You can add publications to the reference database to define sets of data that can be synchronized separately. If you do not use publications to define which changes are to be synchronized, all changes are synchronized.

Publications are used for several purposes in SQL Anywhere. A publication consists of a set of articles. In general, each article can be a whole table, or can define a subset of the data in a table.

Articles defined for UltraLite applications can use row subsets by supplying a WHERE clause, but cannot use column subsets or the SUBSCRIBE BY clause. Articles in UltraLite publications governing HotSync synchronization cannot use a WHERE clause.

❖ To synchronize subsets of data from an UltraLite database

1. Create publications representing the data you wish to synchronize.

   ☞ For more information, see "Creating publications" on page 281.

2. Run the UltraLite generator, specifying the publications on the -v command-line option.

   ☞ For more information, see "The UltraLite Generator" [*UltraLite Database User's Guide*, page 89].

3. When calling the synchronization function, specify the publication.

   If you specify no publication, all changes to the database are synchronized. If you specify one or more publications, only changes that fall within one or more of the listed publications are synchronized.

   ☞ For more information, see the following:

   - **MobileVB** See "ULSyncParms class" [*UltraLite for MobileVB User's Guide*, page 141].
   - **ActiveX** See "ULSyncParms class" [*UltraLite ActiveX User's Guide*, page 134].
   - **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.

- **Embedded SQL and Static C++ API** See "publication synchronization parameter" [*UltraLite C/C++ User's Guide*, page 432].
- **Static Java API** See "publication synchronization parameter" [*UltraLite Static Java User's Guide*, page 78].
- **UltraLite for M-Business Anywhere** See "Class SyncParms" [*UltraLite for M-Business Anywhere User's Guide*, page 112].

Creating publications

Components

Publications can be added to the UltraLite database using the schema painter, the ULXML utility, or from a reference database.

❖ To publish data from an UltraLite database (Schema Painter)

1. Connect to the UltraLite database.
2. In the left pane, open the Synchronization folder.
3. Double-click Add Publication.
4. Specify a set of tables to include in the publication.
5. Click OK to save the changes.

Reference database

For UltraLite synchronization, each article in a publication may include either a complete table or may include a WHERE clause.

❖ To publish data from an UltraLite reference database (Sybase Central)

1. Connect to the database as a user with DBA authority.
2. Open the Publications folder and double-click Add Publication.
3. Type a name for the new publication. Click Next.
4. On the Tables tab, select a table from the list of Matching Tables. Click Add. The table appears in the list of Selected Tables on the right.
5. Add additional tables as required. The order of the tables is not important.
6. If necessary, click the Where tab to specify the rows to be included in the publication. You cannot specify column subsets. If you are using HotSync synchronization, do not specify a WHERE clause.
7. Click Finish.

❖ **To publish data from an UltraLite reference database (SQL)**

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

   ☞ For more information, see "CREATE PUBLICATION statement" [*ASA SQL Reference*, page 385].

Synchronizing high-priority changes

Publications permit the synchronization of specific portions of your UltraLite database. You can combine publications with upload-only or download-only synchronization to synchronize high-priority changes efficiently. Both upload-only and download-only synchronization are less time-consuming than two-way synchronization.

☞ For more information, see " Choosing data to synchronize" on page 280, and the following:

- **MobileVB** See "ULSyncParms class" [*UltraLite for MobileVB User's Guide*, page 141].
- **ActiveX** See "ULSyncParms class" [*UltraLite ActiveX User's Guide*, page 134].
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.
- **Embedded SQL and Static C++ API** See "upload_only synchronization parameter" [*UltraLite C/C++ User's Guide*, page 445].
- **Static Java API** See "upload_only synchronization parameter" [*UltraLite Static Java User's Guide*, page 85].
- **UltraLite for M-Business Anywhere** See "Class SyncParms" [*UltraLite for M-Business Anywhere User's Guide*, page 112].

Including non-synchronizing tables in UltraLite databases

By default, all tables in an UltraLite database are synchronized to the consolidated database. You can include tables in your UltraLite database that are excluded from synchronization, but you must explicitly identify these tables when you create your reference database.

Tables with names ending in nosync are excluded from synchronization. You can use these tables for persistent data that is not related to the consolidated database. Other than being excluded from synchronization, you can use these tables in exactly the same way as other tables in the UltraLite database.

You can alternatively use publications to achieve the same effect. For more information, see "Designing sets of data to synchronize separately" on page 280.

Using client-specific data to control synchronization

Some UltraLite applications require client-specific data that control synchronization, but which are not needed on the consolidated database. For example, you may wish your UltraLite applications to indicate which of a number of channels or topics they are interested in, and use this information to download the appropriate rows.

If you create a table in your UltraLite database with a name ending in allsync, all rows of that table are synchronized at each synchronization, whether or not they have been changed since the last synchronization.

You can store user-specific or client-specific data in allsync tables. If you upload the data in the table to a temporary table in the consolidated database on synchronization, you can use the data to control synchronization by your other scripts without having to be maintained in the consolidated database.

Including read-only tables in an UltraLite database

Some applications include tables in the UltraLite database that are not updated locally. Price lists and company policies are two examples. You can synchronize these tables efficiently by including them in a publication, and synchronizing the publication using download-only synchronization. Download-only synchronization is less time-consuming than a two-way synchronization, as no data is uploaded.

To use download-only synchronization, you must ensure that the data is not changed locally. If any data is changed locally, synchronization fails with a SQLE_DOWNLOAD_CONFLICT error.

Unlike two-way synchronization, you do not have to commit all changes to the UltraLite database before download-only synchronization. Uncommitted changes to tables not involved in synchronization are not uploaded, and so there incomplete transactions do not cause problems.

☞ For information about download-only synchronization, see the following:

♦ **MobileVB** See "ULSyncParms class" [*UltraLite for MobileVB User's Guide*, page 141].

- **ActiveX** See "ULSyncParms class" [*UltraLite ActiveX User's Guide*, page 134].
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.
- **Embedded SQL and Static C++ API** See "download_only synchronization parameter" [*UltraLite C/C++ User's Guide*, page 423].
- **Static Java API** See "download_only synchronization parameter" [*UltraLite Static Java User's Guide*, page 71].
- **UltraLite for M-Business Anywhere** See "Class SyncParms" [*UltraLite for M-Business Anywhere User's Guide*, page 112].

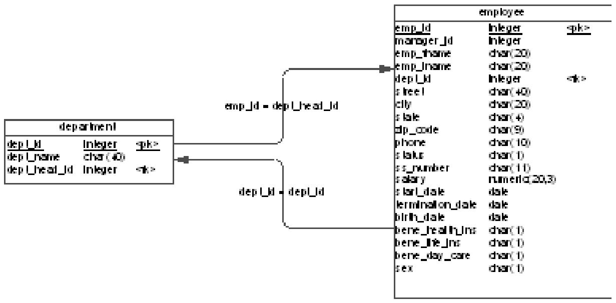
Foreign key cycles

This section describes a specific limitation in UltraLite synchronization that results from a series of tables linked together by foreign keys so that a cycle is formed.

MobiLink synchronization from an UltraLite remote database requires that all changes be committed to the consolidated database in one transaction. To facilitate this single transaction for multiple tables, the inserts, updates, and deletes for each table must be ordered so that operations for a primary table come before the associated foreign table. This ensures that the insert in the foreign table will have its foreign key referential integrity constraint satisfied (likewise for other operations like delete).

The UltraLite analyzer automatically orders all the tables in the remote database so those primary tables are uploaded before foreign tables based on the schema in the reference database. The ordering is always possible as long as there are no foreign key cycles in the schema.

The figure illustrates a simple foreign key cycle between two tables.



If a foreign key cycle is detected by the UltraLite analyzer, the cycle must be broken for the analyzer to successfully complete without any errors. The foreign key cycle must be broken on both the reference database and the consolidated database in order for synchronization transactions to be successfully applied.

For an Adaptive Server Anywhere consolidated and reference database, one of the foreign keys can be made to **check on commit** so that foreign key referential integrity is checked during the commit phase rather than when the operation is initiated. Other database vendors may have similar methods but if not, the schema must be redesigned to eliminate the foreign key cycle.

Example

```
create table c (
    id integer not null primary key,
    c_pk integer not null
);
create table p (
    pk integer not null primary key,
    c_id integer not null,
    foreign key p_to_c (c_id) references c(id)
);
alter table c
add foreign key c_to_p (c_pk)
references p(pk)
check on commit;
```

UltraLite network protocols

Each UltraLite database that synchronizes with a MobiLink synchronization server does so over a network protocol. The network protocol is specified in the UltraLite application. Available network protocols include TCP/IP, HTTP, and HTTPS for TCP/IP-based networks. Support is also provided for HotSync synchronization on the Palm Computing Platform and for ActiveSync synchronization on Windows CE.

☞ For more information, see "Selecting a network protocol" on page 288.

Supported network protocols

UltraLite databases can synchronize with a MobiLink synchronization server over one of a set of network protocols, including TCP/IP, HTTP, and HTTPS. ActiveSync synchronization is available for Windows CE applications under some development models. HotSync is available for Palm OS applications.

Each network protocol has a set of appropriate protocol options. These options set required values for the protocol, such as the location of the MobiLink synchronization server, as well as network-specific control parameters.

The following network protocols are supported for synchronizing UltraLite clients:

Component	TCP/IP	HTTP	HTTPS	ActiveSync (Windows CE only)	HotSync (Palm OS only)
UltraLite ActiveX	✔	✔	✔		
UltraLite for MobileVB	✔	✔	✔[1]		✔
Native UltraLite for Java	✔	✔	✔	✔	
UltraLite.NET	✔	✔	✔	✔	
UltraLite C++	✔	✔	✔	✔	✔
Embedded SQL	✔	✔	✔	✔	✔
Static C++ API	✔	✔	✔	✔	✔
Static Java API	✔[2]	✔	✔		

[1] Must be separately ordered
[2] Use separate protocols for secure synchronization

Separately licensable option required
Transport-layer security requires that you obtain the separately-licensable SQL Anywhere Studio security option and is subject to export regulations.

☞ To order this component, see “Separately-licensable components” [*Introducing SQL Anywhere Studio*, page 5].

Selecting a network protocol

Each network protocol has a set of options that govern its behavior. You should set these synchronization protocol options when you select a network protocol.

The way to select a network protocol and its associated synchronization protocol options depends on the particular UltraLite development model you are using.

♦ For UltraLite for MobileVB and UltraLite ActiveX, the network protocol is one of the synchronization parameters set in the Stream property of the ULSyncParms object. The protocol options are provided as a set of keyword-value pairs in the StreamParms property.

  ☞ For more information, see “ULSyncParms class” [*UltraLite for MobileVB User’s Guide*, page 141], and “ULSyncParms class” [*UltraLite ActiveX User’s Guide*, page 134].

♦ For Native UltraLite for Java applications, the network protocol is set by the **setStream** method of the **SyncParms** object.

  ☞ For more information, see **ianywhere.native_ultralite.SyncParms** in the API Reference.

♦ For embedded SQL and static C++ API applications, the network protocol is set in the stream member of the ul_synch_info structure. The protocol options are supplied in the stream_parms member of the ul_synch_info structure, as a string. The following code is an example for TCP/IP synchronization:

```
ul_synch_info info;
...
info.stream = ULSocketStream();
info.stream_parms = UL_TEXT( "host=myserver" );
```

  ☞ For more information, see the following:

  - “stream synchronization parameter” [*UltraLite C/C++ User’s Guide*, page 438]
  - “stream_parms synchronization parameter” [*UltraLite C/C++ User’s Guide*, page 443]

♦ For static Java applications, the protocol options are supplied using the **setStreamParms** method. The following example illustrates how to call the method:

```
UlSynchOptions synch_options = new UlSynchOptions();
synch_opts.setStream( new UlSocketStream() );
synch_opts.setStreamParms( "host=myserver;port=2439" );
```

☞ For more information, see "stream synchronization parameter" [*UltraLite Static Java User's Guide,* page 81] and "stream_parms synchronization parameter" [*UltraLite Static Java User's Guide,* page 83].

Calling the synchronization function

In order to synchronize data with a MobiLink synchronization server, UltraLite applications call a synchronization function. The particular function depends on the development model you are using and on the network protocol you have selected.

It is helpful to distinguish the following kinds of netowrk protocol:

♦ **Externally-initiated synchronization streams** ActiveSync and HotSync synchronization are initiated by an external application.

☞ For information about calling HotSync synchronization, see the following:

- **MobileVB** See "Synchronizing data" [*UltraLite for MobileVB User's Guide,* page 32].
- **Embedded SQL and Static C++ API** See "Adding HotSync synchronization to Palm applications" [*UltraLite C/C++ User's Guide,* page 125].

♦ **Direct network protocols** TCP/IP, HTTP, and HTTPS network protocols are initiated directly from UltraLite.

☞ For information about calling the synchronization function for these protocols, see the following:

- **UltraLite for MobileVB** See "Synchronize method" [*UltraLite for MobileVB User's Guide,* page 100].
- **UltraLite ActiveX** See "Synchronize method" [*UltraLite ActiveX User's Guide,* page 97].
- **Native UltraLite for Java** See **ianywhere.native_ultralite.Connection.synchronize** in the API Reference.
- **UltraLite.NET** See Synchronize in the UltraLite.NET API Reference.

- **Embedded SQL** See “ULSynchronize function” [*UltraLite C/C++ User’s Guide,* page 388].
- **Static C++ API** See “Synchronize method” [*UltraLite C/C++ User’s Guide,* page 317].
- **Static Java API** See “synchronize method” [*UltraLite Static Java User’s Guide,* page 59].

Maintaining primary key uniqueness

You can declare the default value of a column in a reference database to be of type GLOBAL AUTOINCREMENT. You can use this default for any column in which you want to maintain unique values, but it is particularly useful for primary keys. This feature simplifies the task of generating unique values in setups where data is being replicated among multiple databases, typically by MobiLink synchronization.

When you specify default global autoincrement, the domain of values for that column is partitioned. Each partition contains the same number of values. For example, if you set the partition size for an integer column in a database to 1000, one partition extends from 1001 to 2000, the next from 2001 to 3000, and so on.

☞ For information about declaring columns as global autoincrement in your reference database, see "Declaring default global autoincrement columns" on page 291.

To use global autoincrement columns in your UltraLite database, you must first assign each copy of the database a unique global database identification number. UltraLite then supplies default values for the column only from the partition uniquely identified by that database's number. For example, if you assigned a database in the above example the identity number 1, the default values in that database would be chosen in the range 1001–2000. Another copy of the database, assigned the identification number 2, would supply default value for the same column in the range 2001–3000.

☞ For information about assigning global database identification numbers, see "Setting the global database identifier" on page 292.

☞ For more information about using global autoincrement values in MobiLink remote databases, see "Maintaining unique primary keys using global autoincrement" [*MobiLink Administration Guide,* page 57].

Declaring default global autoincrement columns

You declare default column values in the Adaptive Server Anywhere reference database. When you build your UltraLite application, your UltraLite database inherits the default column value. You can set default values in your reference database by selecting the column properties in Sybase Central, or by including the DEFAULT GLOBAL AUTOINCREMENT phrase in a TABLE or ALTER TABLE statement.

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

For columns of type INT or UNSIGNED INT, the default partition size is 2^{16} = 65536; for columns of other types the default partition size is 2^{32} = 4294967296. Since these defaults may be inappropriate it is best to specify the partition size explicitly.

For example, the following statement creates a simple reference table with two columns: an integer that holds a customer identification number and a character string that holds the customer's name.

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

In the above example, the chosen partition size is 5000.

Default partition sizes for some data types are different in UltraLite applications than in Adaptive Server Anywhere databases. Declare the partition size explicitly if you wish the reference database to behave in the same manner as your UltraLite application.

☞ For more information on GLOBAL AUTOINCREMENT, see "CREATE TABLE statement" [*ASA SQL Reference,* page 407].

Setting the global database identifier

When deploying an application, you must assign a different identification number to each database. You can accomplish the task of creating and distributing the identification numbers by a variety of means. One method is to place the values in a table and download the correct row to each database based on some other unique property, such as user name.

The method of setting this identification number varies according to the programming interface you are using.

- ♦ **UltraLite for MobileVB** See "Properties" [*UltraLite for MobileVB User's Guide,* page 91].
- ♦ **UltraLite ActiveX** See "Properties" [*UltraLite ActiveX User's Guide,* page 91].
- ♦ **Native UltraLite for Java** See **ianywhere.native_ultralite.Connection.databaseID** in the API Reference.
- ♦ **UltraLite.NET** See Connection in the UltraLite.NET API Reference.

- **Embedded SQL** See "ULSetDatabaseID function" [*UltraLite C/C++ User's Guide,* page 385].
- **Static C++ API** See "SetDatabaseID method" [*UltraLite C/C++ User's Guide,* page 316].
- **Static Java API** See "setDatabaseID method" [*UltraLite Static Java User's Guide,* page 59].
- **UltraLite for M-Business Anywhere** See "Method setDatabaseID" [*UltraLite for M-Business Anywhere User's Guide,* page 69].

How default values are chosen

The global database identifier in each deployed UltraLite application must be set to a unique, non-negative integer before default values can be assigned. These identification numbers uniquely identify the databases.

☞ For information, see "Setting the global database identifier" on page 292.

The range of default values for a particular database is $pn + 1$ to $p(n + 1)$, where p is the partition size and n is the global database identification number. For example, if the partition size is 1000 and the global database identification number is set to 3, then the range is from 3001 to 4000.

UltraLite applications choose default values by applying the following rules:

- If the column contains no values in the current partition, the first default value is $pn + 1$.
- If the column contains values in the current partition, but all are less than $p(n + 1)$, the next default value will be one greater than the previous maximum value in this range.
- Default column values are not affect by values in the column outside of the current partition; that is, by numbers less than $pn + 1$ or greater than $p(n + 1)$. Such values may be present if they have been replicated from another database via MobiLink synchronization.

Caution
Column values downloaded via MobiLink synchronization do not update the default value counter. Thus, an error can occur should one MobiLink client insert a value into another client's partition. To avoid this problem, ensure that each copy of your UltraLite application inserts values only in its own partition.

If the global database identification number is set to the default value of 2147483647, a NULL value is inserted into the column. Should NULL values not be permitted, the attempt to insert the row causes an error. This situation arises, for example, if the column is contained in the table's primary key.

Because the global database identification number cannot be set to negative values, the values chosen are always positive. The maximum identification number is restricted only by the column data type and the partition size.

Null default values are also generated when the supply of values within the partition has been exhausted. In this case, a new global database identification number should be assigned to the database to allow default values to be chosen from another partition. Attempting to insert the NULL value causes an error if the column does not permit nulls.

Should the values in a particular partition become exhausted, you can assign a new database identification number to that database. You can assign new database id numbers in any convenient manner. However, one possible technique is to maintain a pool of unused database id values. This pool is maintained in the same manner as a pool of primary keys.

☞ For information about determining whether the range of default values is becoming exhausted, see "Detecting the number of available default values" on page 295.

☞ For information about maintaining primary key uniqueness using explicit primary key pools, see "Maintaining unique primary keys" [*MobiLink Administration Guide*, page 56].

Determining the most recently assigned value

You can retrieve the value that was chosen during the most recently insert operation. Since these values are often used for primary keys, knowing the generated value may let you more easily insert rows that reference the primary key of the first row.

From embedded SQL, you can obtain the most recently assigned global autoincrement default value using the following statement.

```
select @@identity
```

From the C++ API, the value is available using the **GetLastIdentity()** method on the **ULConnection** object

The returned value is an unsigned 64-bit integer, database data type UNSIGNED BIGINT. Since this statement only allows you to determine the most recently assigned default value, you should retrieve this value soon after executing the insert statement to avoid spurious results.

Occasionally, a single insert statement may include more than one column of type global autoincrement. In this case, the return value is one of the generated default values, but there is no reliable means to determine which one. For this reason, you should design your database and write your insert statements so as to avoid this situation.

Detecting the number of available default values

Default values are chosen from the partition identified by the global database identification number until the maximum value is reached. When this state has been reached or is imminent, you must assign the database a new identification number.

The programming interfaces provide means of obtaining the proportion of numbers that have been used. The return value is a short in the range 0–100 that represents the percent of values used thus far. For example, a value of 99 indicates that very few unused values remain and the database should be assigned a new identification number.

The method of setting this identification number varies according to the programming interface you are using.

- **UltraLite for MobileVB** See "Properties" [*UltraLite for MobileVB User's Guide,* page 91].
- **UltraLite ActiveX** See "Properties" [*UltraLite ActiveX User's Guide,* page 91].
- **Native UltraLite for Java** See **ianywhere.native_ultralite.Connection.databaseID** in the API Reference.
- **UltraLite.NET** See Connection in the UltraLite.NET API Reference.
- **Embedded SQL** See "ULGlobalAutoincUsage function" [*UltraLite C/C++ User's Guide,* page 372].
- **Static C++ API** See "GlobalAutoincUsage method" [*UltraLite C/C++ User's Guide,* page 311].
- **Static Java API** See "globalAutoincUsage method" [*UltraLite Static Java User's Guide,* page 58].
- **UltraLite for M-Business Anywhere** See "Method getGlobalAutoIncrementUsage" [*UltraLite for M-Business Anywhere User's Guide,* page 67].

Synchronizing UltraLite databases on the Palm Computing Platform

This section describes the details of synchronization that are specific to the Palm Computing Platform.

☞ For more information about UltraLite on the Palm Computing Platform, see "Developing UltraLite Applications for the Palm Computing Platform" [*UltraLite C/C++ User's Guide*, page 113].

Choosing a synchronization method

Synchronization on the Palm Computing Platform can be carried out using HotSync or over standard network protocols using TCP/IP or HTTP. Each synchronization method has its advantages and disadvantages.

♦ **Multiple applications** If you have more than one UltraLite application installed on a Palm device, they all synchronize when you invoke HotSync. To synchronize multiple applications through a TCP/IP or HTTP connection, you must activate and synchronize each application in turn.

♦ **Universal Serial Bus support** HotSync synchronization has automatic support for USB.

♦ **Publications** Synchronization using HotSync cannot include WHERE clauses.

☞ For more information, see "Designing sets of data to synchronize separately" on page 280.

Understanding HotSync synchronization

UltraLite applications on Palm devices can synchronize over a TCP/IP or HTTP protocol, in much the same manner as UltraLite applications on other platforms. They can also synchronize using the Palm-specific HotSync protocols, which operate in a different manner. This section describes the architecture of the HotSync synchronization.

The sequence of events that occur during HotSync synchronization is as follows:

1. When your UltraLite application is closed, it saves the state of your UltraLite application. The state information is stored in the Palm database, separately from the UltraLite database.

☞ For more information, see “Saving state in UltraLite Palm applications” [*UltraLite C/C++ User’s Guide*, page 120].

2. When you synchronize your Palm device, HotSync calls the MobiLink conduit to synchronize with the MobiLink synchronization server. The MobiLink conduit reads the pages from the UltraLite database and sends the upload to the MobiLink synchronization server.

3. The MobiLink synchronization server integrates updates into the consolidated database and sends a download stream to the conduit.

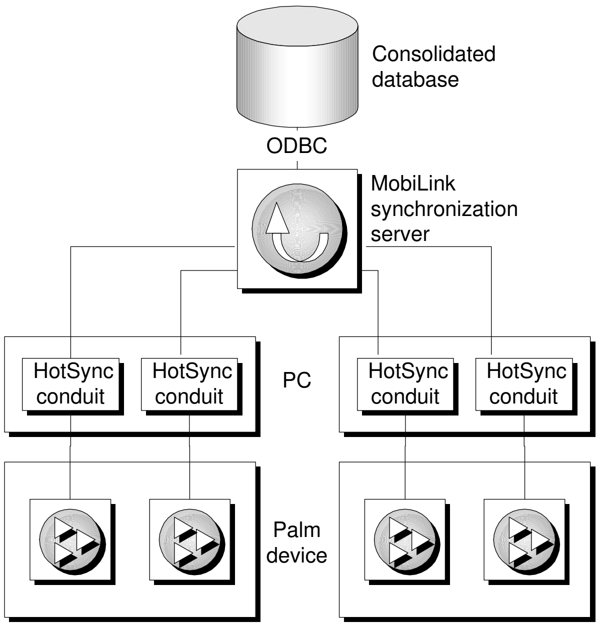
4. The conduit integrates the download stream into the UltraLite database on the Palm device.

5. When your application is launched, it loads the previously saved state of your UltraLite application.

   ☞ For more information, see “Restoring state in UltraLite Palm applications” [*UltraLite C/C++ User’s Guide*, page 121].

HotSync architecture

The following diagram depicts the HotSync architecture. A separate HotSync conduit is required for each application. You can have multiple HotSync conduits on a single PC.



☞ For a description of how to set up your MobiLink HotSync conduit, see “Configuring the MobiLink HotSync conduit” on page 301.

HotSync configuration overview

During HotSync synchronization, the HotSync Manager starts the MobiLink HotSync conduit, *dbhsync9.dll*, which reads from the device and then sends the upload stream to a MobiLink synchronization server. It then receives the download stream from the MobiLink synchronization server and writes the download to the device.

The MobiLink HotSync conduit synchronizes with the MobiLink synchronization server using one of TCP/IP, HTTP, or HTTPS protocols.

In most applications, only the MobiLink HotSync conduit is deployed onto the desktop machines of users.

☞ For information about HotSync architecture, see "Understanding HotSync synchronization" on page 296.

❖ To install and configure the MobiLink HotSync conduit

1. Place the MobiLink conduit files on the user's machine.

   ☞ For instructions, see "Conduit files" on page 298.

2. Register the MobiLink conduit to the HotSync Manager. The HotSync Manager is then able to use the MobiLink conduit.

   ☞ For instructions, see "Registering the MobiLink HotSync conduit to HotSync Manager" on page 299.

3. If you did not include a **stream_parms** parameter in your UltraLite **ul_synch_info** structure, enter these parameters from the HotSync Manager.

   ☞ For instructions, see "Configuring the MobiLink HotSync conduit" on page 301.

   ☞ For information about including **stream_parms** parameter in your UltraLite synchronization call, see "Adding HotSync synchronization to Palm applications" [*UltraLite C/C++ User's Guide*, page 125].

4. If you are using an encrypted database, enter the encryption key in the conduit configuration dialog. If you do not enter this key, you will have to enter it on every synchronization.

   ☞ For instructions, see "Configuring the MobiLink HotSync conduit" on page 301.

Conduit files

The HotSync conduit consists of the following files:

♦ **dbhsync9.dll** The DLL that is called by the HotSync Manager.

- **dblgen9.dll** The language resource library. For languages other than English, the letters en in the file name are replaced by a two-letter abbreviation for the language, such as *dblgde9.dll* or *dblgja9.dll*.
- **Stream DLL** You need a DLL for the communication between the conduit and the MobiLink synchronization server. A separate DLL is provided for each network protocol:
  - For TCP/IP, use *dbmlsock9.dll*.
  - For HTTP, use *dbmlsock9.dll* and *dbmlhttp9.dll*.
  - For HTTPS, use *dbmlhttps9.dll*
  - If you use encryption for this communication, you also need to supply the encryption DLL *dbmltls9.dll*.

These files should be in the same directory, in your system path. When you install SQL Anywhere Studio, they are installed into the operating system subdirectory of your installation directory, which is already in the system path. However, you do not have to install SQL Anywhere Studio to use these files.

Registering the MobiLink HotSync conduit to HotSync Manager

UltraLite includes a command line **conduit installation utility** named *dbcond9* to make a set of registry entries for the HotSync Manager to be able to use the MobiLink conduit. This utility requires the following files:

- *dbcond9.exe*
- *condmgr.dll*

❖ To deploy the conduit installation utility

1. Choose a top-level deployment directory.

   For example, you may choose a directory named *c:\deploy*.

2. Add a registry entry with the full path of the deployment directory as its value.

   The registry entry must be as follows:

   ```
   HKEY_CURRENT_USER\Software\Sybase\Adaptive Server Anywhere\
             version-string\Location
   ```

   where *version-string* is a number representing your version of the SQL Anywhere Studio (such as **9.0**). If the entry is not found in *HKEY_CURRENT_USER*, the software looks in *HKEY_LOCAL_MACHINE*.

3. Add the *dbcond9.exe* file to the *win32* subdirectory of the deployment directory.

4. Add the *condmgr.dll* file.

   The *condmgr.dll* file must go in the *win32\condmgr* subdirectory of the deployment directory.

The SQL Anywhere Studio installation creates the required registry entries and places files in the appropriate locations.

❖ To register the MobiLink HotSync conduit to HotSync Manager

1. Ensure that the HotSync conduit files and the files for the conduit installation utility are in place.

2. Run the conduit installation utility. On the command line, you must specify the creator ID of the Palm application and a name that HotSync will use to identify the conduit.

   For example, the following command installs a conduit for the application with creator ID **Syb2**, named **CustDB**. These are the settings for the CustDB sample application:

   ```
   dbcond9 "Syb2" -n CustDB
   ```

   ☞ For more information about the conduit installation utility, see "The HotSync Conduit Installer" [*UltraLite Database User's Guide,* page 99].

Note A secondary location for HotSync synchronization depends on the version of the Palm Computing Platform software you are using. This secondary location may be under the *HKEY_CURRENT_USER\Software\U.S. Robotics* or *HKEY_CURRENT_USER\Software\Palm Computing* folders.

Checking that MobiLink HotSync conduit installation is correct

Following are instructions for verifying that your conduit is installed and is working.

❖ To check that the HotSync conduit is properly installed

1. Check that a conduit is installed:
   - ♦ In your PC's system tray, right-click HotSync Manager.
   - ♦ From the pop-up menu, choose Custom.

     A list of conduits is displayed for each user. Verify that your conduit is listed.

2. Set the system environment variable UL_DEBUG_CONDUIT to any value.
3. Shut down and restart the HotSync Manager.
4. If the MobiLink conduit is properly installed, two dialog boxes appear. If no dialog appears, the conduit was not properly installed.
5. Unset the environment variable.
6. Shut down and restart the HotSync Manager.

MobiLink must be started before using HotSync
Before using HotSync, the MobiLink synchronization server must be started and be ready to accept connections from the MobiLink HotSync conduit. The MobiLink synchronization server does not have to be on the same computer, but it must be reachable across the network.

Configuring the MobiLink HotSync conduit

The MobiLink HotSync conduit needs to communicate with a MobiLink synchronization server to synchronize the UltraLite application and the consolidated database. You can provide the information needed by the conduit to locate the MobiLink synchronization server in a **stream_parms** member of the UltraLite **ul_synch_info** structure supplied to the ULSetSynchInfo function. If you did not specify a **stream_parms** value, or if you specified the value as null, you can enter the required parameters from the HotSync Manager.

In addition, if you are using a strongly encrypted UltraLite database, you can save the encryption key so that you do not have to enter it on each synchronization.

If you have Palm Desktop software installed, the Adaptive Server Anywhere installation creates registry entries for the **CustDB** sample application. You can use these entries as a starting point for your own application.

☞ For information about **stream_parms**, see "Adding HotSync synchronization to Palm applications" [*UltraLite C/C++ User's Guide*, page 125].

❖ To configure the MobiLink HotSync conduit for synchronization

1. Right-click the HotSync Manager icon in the system tray, and choose Custom from the popup menu.
2. Select your MobiLink HotSync conduit from the list of conduit names, and click Change.

3. Enter a set of network protocol options in the Synchronization Parameters text box. These options are the same as those in a **stream_parms** parameter, except that a “stream” entry is used to specify the network protocol type (TCPIP, HTTP, or HTTPS). For example:

   ```
   stream=tcpip;host=localhost
   ```

   ☞ For more information, see “HotSync protocol options” on page 343.

4. If the database is strongly encrypted, you can enter the encryption key in the Encryption Key text box. If no key is entered, you will be prompted for the encryption key on each synchronization.

5. Click OK to complete the entry. The HotSync conduit is now ready to use.

Registry locations

The protocol options and encryption key are stored in the registry in *HKEY_CURRENT_USER\Software\Sybase\Adaptive Server Anywhere\9.0\Conduit\ Creator-ID*, where *Creator-ID* is application-dependent.

A secondary location for HotSync synchronization depends on the version of the Palm Computing Platform software you are using. They are made under the *HKEY_CURRENT_USER\Software\U.S. Robotics* or the *HKEY_CURRENT_USER\Software\Palm Computing* folder.

HotSync log files

HotSync records when each synchronization took place and whether each installed conduit worked as expected. The HotSync log file is in the subdirectory *User\HotSync.log* of your Pilot or Palm directory.

You can obtain additional debugging information in your HotSync log file by setting the UL_DEBUG_CONDUIT_LOG environment variable. This variable is useful during development if you have problems with the HotSync conduit. By default, the environment variable is not set.

UL_DEBUG_CONDUIT_LOG = 1 When set to 1, basic information is written to the HotSync log file, such as synchronization parameters, registry locations, and attempts to load libraries.

UL_DEBUG_CONDUIT_LOG = 2 When set to 2, more detailed information is written to the HotSync log file.

You must restart HotSync before the new setting takes effect.

☞ For information about how to set environment variables, see “Setting environment variables” [*ASA Database Administration Guide*, page 276].

Deploying the MobiLink HotSync conduit

For applications using HotSync synchronization, each end user must have the MobiLink HotSync conduit installed on their desktop. This installation requires the following:

- ♦ **Deploy the conduit files** The files for the conduit must be installed into a location in the end user's system path.

  ☞ For a list of conduit files, see "Conduit files" on page 298.

- ♦ **Install the conduit** You can deploy the conduit installation utility to your end users and provide instructions for them to run it, or you can use the HotSync Manager to install the conduit.

  ☞ For instructions, see "Registering the MobiLink HotSync conduit to HotSync Manager" on page 299.

- ♦ **Configure the conduit** If you did not include a **stream_parms** parameter in your UltraLite **ul_synch_info** structure, enter these parameters from the HotSync Manager. Also, if you are using an encrypted database, you may want to enter the encryption key.

  ☞ For instructions, see "Configuring the MobiLink HotSync conduit" on page 301.

Configuring TCP/IP, HTTP, or HTTPS synchronization

This section describes how to configure the synchronization setup for UltraLite Palm applications using TCP/IP or HTTP synchronization.

☞ For information about synchronization architecture for HTTP or TCP/IP communications, see "Parts of the synchronization system" [*MobiLink Administration Guide*, page 8].

Configuring TCP/IP synchronization for the Palm Computing Platform

There are two ways of using TCP/IP networking in a Palm device. In either case, you must connect to a Remote Access Service (RAS). The difference lies in how you make the connection to the RAS.

- ♦ **Use a modem to dial into an ISP** The Internet Service Provider (ISP) must provide access to a Remote Access Service (RAS). The components of the connection are as follows:

```
Application
<--> Palm Net Library
<--> Palm modem
<--> NT RAS
<--> TCP/IP network
```

- ♦ **Connect via the serial port to a Windows NT machine** The components of the connection are as follows:

```
Application
<--> Palm Net Library
<--> serial cable
<--> NT RAS
<--> TCP/IP network
```

When using TCP/IP, the MobiLink synchronization server can be any machine on the network that is accessible via TCP/IP.

Before synchronization, the following conditions must be satisfied:

1. The device must be in its cradle.

2. If you are using the serial port to connect to a Windows NT machine running RAS, the HotSync Manager and other applications that use the serial port must be shut down. Windows NT only allows one application to use a serial port at a time.

3. The MobiLink synchronization server must be started. By default, the MobiLink synchronization server listens for TCP/IP communications over port 2439.

4. The Palm device must have Network settings in place so that it can connect to the network. Modem settings are also required if using a modem to dial into an ISP.

Configuring HTTP or HTTPS synchronization for the Palm Computing platform

To use HTTP or HTTPS synchronization, you must first configure RAS TCP/IP synchronization. For information about configuring RAS, see "Configuring TCP/IP synchronization for the Palm Computing Platform" on page 303.

When using HTTP or HTTPS, the MobiLink synchronization server can be any machine on the network that is accessible via the protocol.

❖ To synchronize using HTTP or HTTPS

1. Place the Palm device in its cradle.

2. If you are using the serial port to connect to a desktop machine running RAS, shut down the HotSync Manager and other applications that use the serial port. Windows only allows one application to use a serial port at a time.

3. Start the MobiLink synchronization server.

4. Ensure that the network settings on the Palm device are configured so that it can connect to the network. Modem settings are also required if using a modem to dial into an ISP.

   ☞ For more information, see "Configuring TCP/IP synchronization for the Palm Computing Platform" on page 303.

Configuring Remote Access Service

Synchronizing Palm applications over TCP/IP, HTTP, or HTTPS requires a Remote Access Service on your desktop machine. Remote Access Service software is not part of UltraLite or MobiLink. Configuring Remote Access Service software is tricky, however, and so this section provides some instructions to assist with the task.

Configuring RAS for synchronization via modem

To use this method, you must have access to a Remote Access Service when you dial in.

❖ To configure a Palm device for RAS TCP/IP via a modem

1. Install the modem by plugging the Palm device into the modem module.

2. Go to the Preferences (Prefs) panel and choose Network from the dropdown list at the top right of screen.

3. Choose the Windows RAS service.

4. Set the dial-in username and password.

5. Set the phone number to the number at which the Remote Access Service can be reached. Obtain this number from your ISP.

6. Tap Details.

7. Set the connection type (usually PPP).

8. Set the DNS and IP addresses as recommended by your network administrator.

9. Tap Script and enter the script recommended by your ISP. This script will be similar to the following sample.

```
Wait For: Username:
Delay: 1
Send UserID:
Send CR:
Wait For: Password:
Delay: 1
Send Password:
Send CR:
Wait For: >
Delay: 1
Send: ppp
Send CR:
End:
```

Tap OK until you are back to the Network Preferences.

At this point, you are ready to test your TCP/IP connection.

Configuring RAS for serial port connection

This procedure involves actions both on Windows NT and on the Palm Computing device.

❖ To configure Windows NT for RAS TCP/IP via serial port

1. From the Control Panel, open Modems. Make sure that a modem is defined for **Dial-Up Networking Serial Cable between 2 PCs** on the COM port to which the cradle is connected.

2. Set the speed for this modem to the baud rate you are using. The default is 19200.

3. Make sure TCP/IP protocol is installed. Select Start ➤ Settings ➤ Control Panel and double-click the Network icon. Click on the Protocols tab. If there is no TCP/IP entry, choose Add to install it.

4. Enable IP Forwarding (in the Routing tab of TCP/IP properties)

5. Under the Services tab, make sure that Remote Access Service is installed. If there is no entry for Remote Access Service, choose Add to install it.

   In Remote Access Service Properties, add **Dial Up Network serial cable between 2 pc's** for that COM port if the cradle's COM port is not in the list of ports.

6. Configure this entry to receive calls. In the RAS Network properties set encryption settings to **Allow any authentication including clear text**. In the RAS Network properties allow only TCP/IP client.

7. Configure TCP/IP. Allow clients to access the entire network. Assigning the TCP/IP addresses depends on your network. Contact your network administrator for details.

8. Add a user for dial-in access. Select Start ➤ Programs ➤ Administrative Tools ➤ User Manager. Uncheck **User Must Change Password at Next Logon**. Choose the Dialin button, and grant dialin permission to user with No Call Back.

9. If the RAS COM port is the same one that HotSync Manager uses, shut down the HotSync Manager or any other applications that use the COM port.

10. Start the Remote Access Administrator. Select Start ➤ Programs ➤ Administrative Tools ➤ Remote Access Admin.

11. Start the RAS service. Select Server ➤ Start Remote Access Service. Choose to start the service on the local machine.

HotSync Manager or any other applications that use the serial port and the RAS service will not run at the same time. One must be shut down first for the other to run, as Windows NT prevents two different applications from accessing the same serial port. You have to stop the RAS service (Server ➤ Stop Remote Access Service from the Remote Access Admin) before you can restart the HotSync Manager. Alternatively, you can use separate serial ports.

Once the RAS service is running, it is ready to receive connection requests via the serial port.

❖ To configure a Palm device for RAS TCP/IP via serial port

1. Go to the Preferences (Prefs) panel and choose Network from the dropdown list at the top right of screen.

2. Choose the Windows RAS service.

3. Set the dial-in username and password.

4. Set the Palm to use the serial port.
   - For Palm OS 3.3 and above, select **Direct serial**.
   - For earlier versions of the Palm OS, set the phone number to **00** (zero zero). This is a special phone number that tells the Palm to use the serial port directly, instead of a modem.

5. Tap Details.

6. Set the connection type (usually PPP).

7. Set the DNS and IP addresses as recommended by your network administrator.

8. Tap Script and enter the following script:

   ```
   Send: CLIENT
   Send CR:
   Delay: 1
   Send: CLIENT
   END
   ```

   Tap OK until you are back to the Network Preferences

At this point, you are ready to test your TCP/IP connection.

Testing and troubleshooting

❖ To test the connection

1. **via modem** Connect the Palm device to the modem and follow the instructions provided by your ISP for connecting to their network. Once connected, tap the Connect button in Prefs ➤ Network on the Palm device.

2. **via serial port** Ensure RAS is running on the Windows NT machine. Place the Palm device in the cradle and connect the cradle to the correct COM port on the Windows NT machine. Tap the Connect button in Prefs ➤ Network on the Palm device.

With TCP/IP, there are two levels of service. At the minimum level, you can connect to another TCP/IP host using an IP number of the following form.

```
NNN.NNN.NNN.NNN
```

At the next level, when a DNS server is properly configured, you are able to connect to another host by name.

```
some_host_machine.any_company.com
```

Having a DNS service is more convenient, since most people are better at remembering a name than a number. As long as you have the minimum TCP/IP service, and an IP number, you can synchronize an UltraLite application using TCP/IP.

There are a number of steps you can take to troubleshoot TCP/IP connections on the Palm device.

- Hitting the scroll down button on the Palm device during the connection phase displays the progress of the connection.

♦ The connection log is accessible from the Network Preferences panel. Choose View Log from the Options menu to see information about the network connection. The log is an interactive utility for controlling and viewing your connection information. Enter ? for help.

♦ There are several tools for testing a TCP/IP connection from the Palm. You can find most of them at the following locations:

```
http://www.roadcoders.com
http://www.palmcentral.com
```

There are also steps you can take for troubleshooting on Windows NT:

♦ In the Remote Access Admin, double-click on the running server.

♦ Select the appropriate port and choose Port Status. The Port Status dialog shows you the Line condition (connected or waiting for a call) and lets you watch the byte counts for both directions.

Synchronizing UltraLite databases on Windows CE

UltraLite applications on Windows CE can synchronize using standard network protocols (TCP/IP, HTTP, or HTTPS). UltraLite applications built using embedded SQL, the static C++ API, or Native UltraLite for Java can also use ActiveSync.

This section describes ActiveSync synchronization.

Installing the MobiLink provider for ActiveSync

Before you register your application for use with ActiveSync, you must install the MobiLink provider for ActiveSync using the installation utility (*dbasinst.exe*).

The MobiLink provider for ActiveSync includes a desktop component and a device component. You must configure the provider for each device that synchronizes through your desktop machine.

When you have installed the MobiLink provider for ActiveSync you must register each application separately. For instructions, see "Registering applications for use with ActiveSync" on page 311.

❖ To install the MobiLink provider for ActiveSync

1. Ensure that you have the ActiveSync software on your machine, and that the Windows CE device is connected.

2. Enter the following command to install the MobiLink provider:

   ```
   dbasinst -k desk-path -v dev-path
   ```

   where *desk-path* is the location of the desktop component of the provider (*dbasdesk.dll*) and *dev-path* is the location of the device component (*dbasdev.dll*).

   If you have SQL Anywhere installed on your machine, *dbasdesk.dll* is in the *win32* subdirectory of your SQL Anywhere directory and *dbasdev.dll* is in a platform-specific directory in the *CE* subdirectory. These directories are default search locations, and you can omit both the -k and -v options.

   ☞ For more information, see "ActiveSync provider installation utility" on page 28.

3. Restart your machine.

ActiveSync does not recognize new providers until the machine is restarted.

4. Enable the MobiLink provider.
   - ♦ From the ActiveSync window, click Options.
   - ♦ Check the MobiLink item in the list and click OK to activate the provider.
   - ♦ To see a list of registered applications, click Options again, choose the MobiLink provider, and click Settings.

     ☞ For more information about registering applications, see "Registering applications for use with ActiveSync" on page 311.

Registering applications for use with ActiveSync

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

☞ For information about the alternative approach, see "ActiveSync provider installation utility" on page 28.

❖ To register an application for use with ActiveSync

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

   ☞ For information, see "Installing the MobiLink provider for ActiveSync" on page 310.

2. Start the ActiveSync software on your desktop machine.

3. From the ActiveSync window, choose Options.

4. From the list of information types, choose MobiLink and click Settings.

5. In the MobiLink Synchronization dialog, click New. The Properties dialog appears.

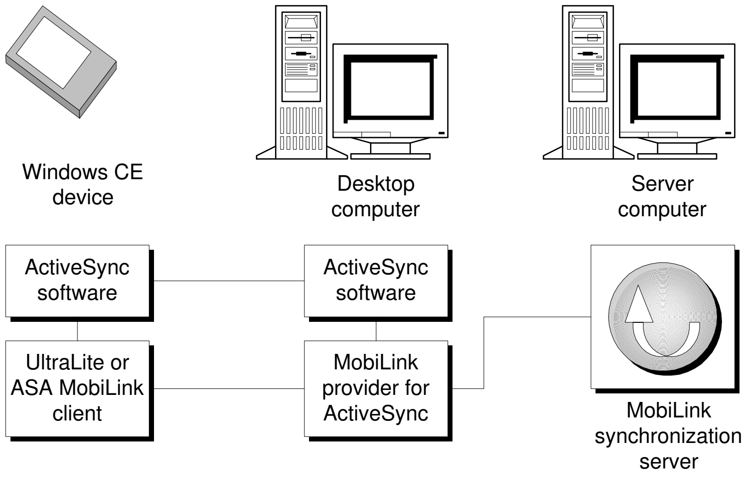
6. Enter the following information for your application:
   - ♦ **Application name** A name identifying the application to be displayed in the ActiveSync user interface.
   - ♦ **Class name** The registered class name for the application.

     ☞ See "Assigning class names for applications" [*UltraLite C/C++ User's Guide*, page 137].
   - ♦ **Path** The location of the application on the device.
   - ♦ **Arguments** Any command line arguments to be used when ActiveSync starts the application.

7. Click OK to register the application.

Deploying applications that use ActiveSync

Applications that use ActiveSync synchronization must be registered with ActiveSync as well as copied onto the device. Also, each desktop machine must have the MobiLink provider for ActiveSync installed. The architecture for ActiveSync is illustrated in the following diagram.



Windows CE device
Desktop computer
Server computer
ActiveSync software
ActiveSync software
UltraLite or ASA MobiLink client
MobiLink provider for ActiveSync
MobiLink synchronization server

❖ To deploy ActiveSync applications

1. Install the MobiLink provider for ActiveSync on each end user's computer.

   An ActiveSync provider install utility is provided with SQL Anywhere. This is the *dbasinst.exe* command line utility.

   ☞ For information about the *dbasinst.exe* command line utility, see "Installing the MobiLink provider for ActiveSync" on page 310 and "ActiveSync provider installation utility" on page 28.

2. Register the application for use with ActiveSync.

   You can register the application either by using ActiveSync, or by using the ActiveSync provider installation utility *dbasinst.exe*.

   ☞ For information about registering the application, see "Registering applications for use with ActiveSync" on page 311.

3. Copy the application onto the device.

If your application is a single executable, statically linked with the runtime library, you can use the ActiveSync provider installation utility *dbasinst.exe* to copy the application to the device.

If the application includes multiple files (for example, if you use the UltraLite runtime DLL rather than the static runtime library), you must copy the files onto the device in some other way.

CHAPTER 12

UltraLite Synchronization Parameters

About this chapter

This chapter details all the parameters that you can set to synchronize your UltraLite applications using MobiLink.

☞ The parameters described in this chapter only apply to UltraLite remote databases. To synchronize Adaptive Server Anywhere remote databases, see "Adaptive Server Anywhere Client Synchronization Parameters" on page 95.

Contents

Synchronization parameters

Synchronization parameters control the synchronization between an UltraLite database and the MobiLink synchronization server. The way you set the parameters depends on the specific UltraLite interface you are using. This section describes the effects of the parameters, and provides links to other locations for information on how to set them.

The Stream Type and Version parameters are required. If they are not set, the synchronization function throws an exception (SQLCode.SQLE_SYNC_INFO_INVALID or its equivalent).

Some synchronization parameters can conflict. Only one of Download Only, Ping, Resume Partial Download, or Upload Only may be specified at a time. If more than one of these parameters is set to true, the synchronization function throws an exception (SQLCode.SQLE_SYNC_INFO_INVALID or its equivalent).

Authentication Parameters synchronization parameter

Function

Supplies parameters to the authentication_parameters script.

Usage

Use this parameter to supply any values required by your authentication_parameters script. These may be a user name and password, for example.

If you use this parameter, you must also supply the number of parameters. See "Number of Authentication Parameters parameter" on page 322.

Allowed values

An array of strings. Null is not allowed as a value for any of the strings, but you can supply an empty string.

See also

"Number of Authentication Parameters parameter" on page 322

"authenticate_parameters connection event" [*MobiLink Administration Guide*, page 334]

Interfaces

- **MobileVB** "ULSyncParms class" [*UltraLite for MobileVB User's Guide*, page 141]
- **ActiveX** "ULSyncParms class" [*UltraLite ActiveX User's Guide*, page 134]
- **UltraLite.NET** See "AuthenticationParms property" [*UltraLite.NET User's Guide*, page 264] (iAnywhere.Data.UltraLite namespace) and "AuthenticationParms property" [*UltraLite.NET User's Guide*, page 492] (iAnywhere.UltraLite namespace).
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.

- **C++ Component, embedded SQL, and Static C++ API** See "auth_parms parameter" [*UltraLite C/C++ User's Guide,* page 418].
- **Static Java API** See "auth_parms parameter" [*UltraLite Static Java User's Guide,* page 68].

Authentication Status synchronization parameter

Function	Reports the status of MobiLink user authentication. The MobiLink synchronization server provides this information to the client.
Usage	If you are implementing a custom authentication scheme, the authenticate_user or authenticate_user_hashed synchronization script must return one of the allowed values of this parameter. The parameter is set by the MobiLink synchronization server, and so is read-only.
Allowed values	The allowed values are held in an interface-specific enumeration. If a custom **authenticate_user** synchronization script at the consolidated database returns a different value, the value is interpreted according to the rules given in "authenticate_user connection event" [*MobiLink Administration Guide,* page 336].
See also	"Authenticating MobiLink Users" on page 9.

Interfaces

- **MobileVB** "ULSyncResult class" [*UltraLite for MobileVB User's Guide,* page 145]
- **ActiveX** "ULSyncResult class" [*UltraLite ActiveX User's Guide,* page 138]
- **UltraLite.NET** See "AuthStatus property" [*UltraLite.NET User's Guide,* page 285] (iAnywhere.Data.UltraLite namespace) and "AuthStatus property" [*UltraLite.NET User's Guide,* page 543] (iAnywhere.UltraLite namespace).
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncResult** in the API Reference.
- **C++ Component, embedded SQL, and Static C++ API** See "auth_status parameter" [*UltraLite C/C++ User's Guide,* page 419].
- **Static Java API** See "auth_status parameter" [*UltraLite Static Java User's Guide,* page 69].
- **UltraLite for M-Business Anywhere** See "Class SyncResult" [*UltraLite for M-Business Anywhere User's Guide,* page 120].

Authentication Value synchronization parameter

Function	Reports results of a custom MobiLink user authentication script. The MobiLink synchronization server provides this information to the client.
Default	The values set by the default MobiLink user authentication mechanism are described in "Authentication Status synchronization parameter" on page 317.
Usage	The parameter is set by the MobiLink synchronization server, and so is read-only.
See also	"authenticate_user connection event" [*MobiLink Administration Guide,* page 336] "authenticate_user_hashed connection event" [*MobiLink Administration Guide,* page 340] "Authentication Status synchronization parameter" on page 317

Interfaces

- ♦ **MobileVB** "ULSyncResult class" [*UltraLite for MobileVB User's Guide,* page 145]
- ♦ **ActiveX** "ULSyncResult class" [*UltraLite ActiveX User's Guide,* page 138]
- ♦ **UltraLite.NET** See "AuthValue property" [*UltraLite.NET User's Guide,* page 285] (iAnywhere.Data.UltraLite namespace) and "AuthValue property" [*UltraLite.NET User's Guide,* page 543] (iAnywhere.UltraLite namespace).
- ♦ **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.
- ♦ **C++ Component, embedded SQL, and Static C++ API** See "auth_value synchronization parameter" [*UltraLite C/C++ User's Guide,* page 420].
- ♦ **Static Java API** See "auth_value synchronization parameter" [*UltraLite Static Java User's Guide,* page 70].
- ♦ **UltraLite for M-Business Anywhere** See "Class SyncResult" [*UltraLite for M-Business Anywhere User's Guide,* page 120].

Checkpoint Store synchronization parameter

Function

Adds additional checkpoints of the database during synchronization to limit database growth during the synchronization process.

The checkpoint operation adds I/O operations for the application and for the Palm conduit and so slows synchronization. The option is most useful for large downloads with many updates. Devices with slow flash memory may not want to pay the performance penalty.

Default — By default, only required checkpointing is done.

Interfaces

- **MobileVB** “ULSyncParms class” [*UltraLite for MobileVB User’s Guide,* page 141]
- **ActiveX** “ULSyncParms class” [*UltraLite ActiveX User’s Guide,* page 134]
- **UltraLite.NET** See “CheckpointStore property” [*UltraLite.NET User’s Guide,* page 265] (iAnywhere.Data.UltraLite namespace) and “CheckpointStore property” [*UltraLite.NET User’s Guide,* page 493] (iAnywhere.UltraLite namespace).
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.
- **C++ Component, embedded SQL, and Static C++ API** See “checkpoint_store synchronization parameter” [*UltraLite C/C++ User’s Guide,* page 421].
- **Static Java API** Not available.
- **UltraLite for M-Business Anywhere** See “Class SyncParms” [*UltraLite for M-Business Anywhere User’s Guide,* page 112].

Disable Concurrency synchronization parameter

Function — Disallow database access from other threads during synchronization.

Default — The parameter is a Boolean value, and by default is false (allowing concurrent database access). Data access is read-write during the download phase, and read-only otherwise.

See also — “Understanding concurrency in UltraLite” [*UltraLite Database User’s Guide,* page 58]

Interfaces

- **MobileVB** Not available
- **ActiveX** Not available
- **UltraLite.NET** See “DisableConcurrency property” [*UltraLite.NET User’s Guide,* page 265] (iAnywhere.Data.UltraLite namespace) and “DisableConcurrency property” [*UltraLite.NET User’s Guide,* page 493] (iAnywhere.UltraLite namespace).
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.

- **C++ Component, embedded SQL, and Static C++ API** See "disable_concurrency synchronization parameter" [*UltraLite C/C++ User's Guide*, page 422].
- **Static Java API** Not available.
- **UltraLite for M-Business Anywhere** See "Class SyncParms" [*UltraLite for M-Business Anywhere User's Guide*, page 112].

Download Only synchronization parameter

Function	Do not upload any changes from the UltraLite database during this synchronization.
Default	The parameter is a Boolean value, and by default is false.
See also	"Including read-only tables in an UltraLite database" on page 283.
	"Upload Only synchronization parameter" on page 337

Interfaces

- **MobileVB** "ULSyncParms class" [*UltraLite for MobileVB User's Guide*, page 141]
- **ActiveX** "ULSyncParms class" [*UltraLite ActiveX User's Guide*, page 134]
- **UltraLite.NET** See "DownloadOnly property" [*UltraLite.NET User's Guide*, page 266] (iAnywhere.Data.UltraLite namespace) and "DownloadOnly property" [*UltraLite.NET User's Guide*, page 494] (iAnywhere.UltraLite namespace).
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.
- **C++ Component, embedded SQL, and Static C++ API** See "download_only synchronization parameter" [*UltraLite C/C++ User's Guide*, page 423].
- **Static Java API** See "download_only synchronization parameter" [*UltraLite Static Java User's Guide*, page 71].
- **UltraLite for M-Business Anywhere** See "Class SyncParms" [*UltraLite for M-Business Anywhere User's Guide*, page 112].

Ignored Rows synchronization parameter

Function	This boolean parameter is set to **true** if any rows were ignored by the MobiLink synchronization server during synchronization because of absent scripts.

The parameter is read-only.

Interfaces

- **MobileVB** "ULSyncResult class" [*UltraLite for MobileVB User's Guide*, page 145]
- **ActiveX** "ULSyncResult class" [*UltraLite ActiveX User's Guide*, page 138]
- **UltraLite.NET** See "IgnoredRows property" [*UltraLite.NET User's Guide*, page 285] (iAnywhere.Data.UltraLite namespace) and "IgnoredRows property" [*UltraLite.NET User's Guide*, page 543] (iAnywhere.UltraLite namespace).
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncResult** in the API Reference.
- **C++ Component, embedded SQL, and Static C++ API** See "ignored_rows synchronization parameter" [*UltraLite C/C++ User's Guide*, page 425].
- **Static Java API** See "ignored_rows synchronization parameter" [*UltraLite Static Java User's Guide*, page 72].
- **UltraLite for M-Business Anywhere** See "Class SyncResult" [*UltraLite for M-Business Anywhere User's Guide*, page 120].

Keep Partial Download synchronization parameter

Function

When a download fails because of a communications error during synchronization, this parameter controls whether UltraLite holds on to the download rather than rolling back the changes.

Default

The default setting is false, indicating that UltraLite rolls back all changes after a failed download.

Interfaces

- **MobileVB** "ULSyncParms class" [*UltraLite for MobileVB User's Guide*, page 141]
- **ActiveX** "ULSyncParms class" [*UltraLite ActiveX User's Guide*, page 134]
- **UltraLite.NET** See "KeepPartialDownload property" [*UltraLite.NET User's Guide*, page 266] (iAnywhere.Data.UltraLite namespace) and "KeepPartialDownload property" [*UltraLite.NET User's Guide*, page 494] (iAnywhere.UltraLite namespace).
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.
- **C++ Component, embedded SQL, and Static C++ API** See "keep_partial_download synchronization parameter" [*UltraLite C/C++ User's Guide*, page 424].

- **Static Java API** Not available.
- **UltraLite for M-Business Anywhere** See "Class SyncParms" [*UltraLite for M-Business Anywhere User's Guide,* page 112].

See also

- "Resuming failed downloads" [*MobiLink Administration Guide,* page 74]
- "Resume Partial Download synchronization parameter" on page 327

New Password synchronization parameter

Function

Sets a new MobiLink password associated with the user name.

Default

The parameter is optional, and is a string.

See also

"Authenticating MobiLink Users" on page 9.

Interfaces

- **MobileVB** "ULSyncParms class" [*UltraLite for MobileVB User's Guide,* page 141]
- **ActiveX** "ULSyncParms class" [*UltraLite ActiveX User's Guide,* page 134]
- **UltraLite.NET** See "NewPassword property" [*UltraLite.NET User's Guide,* page 267] (iAnywhere.Data.UltraLite namespace) and "NewPassword property" [*UltraLite.NET User's Guide,* page 495] (iAnywhere.UltraLite namespace).
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.
- **C++ Component, embedded SQL, and Static C++ API** See "new_password synchronization parameter" [*UltraLite C/C++ User's Guide,* page 426].
- **Static Java API** See "new_password synchronization parameter" [*UltraLite Static Java User's Guide,* page 73].
- **UltraLite for M-Business Anywhere** See "Class SyncParms" [*UltraLite for M-Business Anywhere User's Guide,* page 112].

Number of Authentication Parameters parameter

Function

Supply the number of authentication parameters being passed to the authentication_parameters script.

Default

No parameters supplied.

See also

"Authentication Parameters synchronization parameter" on page 316

"authenticate_parameters connection event" [*MobiLink Administration Guide,* page 334]

Interfaces

- **MobileVB** “ULSyncParms class” [*UltraLite for MobileVB User’s Guide*, page 141]
- **ActiveX** “ULSyncParms class” [*UltraLite ActiveX User’s Guide*, page 134]
- **UltraLite.NET** See “AuthenticationParms property” [*UltraLite.NET User’s Guide*, page 264] (iAnywhere.Data.UltraLite namespace) or “AuthenticationParms property” [*UltraLite.NET User’s Guide*, page 492] (iAnywhere.UltraLite namespace)
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.
- **C++ Component, embedded SQL, and Static C++ API** See “num_auth_parms parameter” [*UltraLite C/C++ User’s Guide*, page 427].
- **Static Java API** See “num_auth_parms parameter” [*UltraLite Static Java User’s Guide*, page 74].
- **UltraLite for M-Business Anywhere** See “Class SyncParms” [*UltraLite for M-Business Anywhere User’s Guide*, page 112].

Observer synchronization parameter

Function

A pointer to a callback function or event handler that monitors synchronization.

See also

“User Data synchronization parameter” on page 338

Interfaces

- **MobileVB** Declare the connection object With Events. See “ULConnection class” [*UltraLite for MobileVB User’s Guide*, page 91]
- **ActiveX** Use CreateObjectWithEvents when opening the DatabaseManager object. See “ULDatabaseManager class” [*UltraLite ActiveX User’s Guide*, page 101]
- **UltraLite.NET** See “ULSyncProgressListener members” [*UltraLite.NET User’s Guide*, page 281] (iAnywhere.Data.UltraLite namespace) and “SyncProgressListener interface” [*UltraLite.NET User’s Guide*, page 539] (iAnywhere.UltraLite namespace).
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncProgressListener** in the API Reference.
- **C++ Component, embedded SQL, and Static C++ API** See “observer synchronization parameter” [*UltraLite C/C++ User’s Guide*, page 428].

- **Static Java API** See "observer synchronization parameter" [*UltraLite Static Java User's Guide,* page 75].

Partial Download Retained synchronization parameter

Function	When a download fails because of a communications error during synchronization, this parameter controls whether UltraLite applied those changes that were downloaded rather than rolling back the changes.
Default	The parameter is set by UltraLite. If there is no partial download retained, the value is false.

Partial downloads are retained only if Keep Partial Download is set to true.

Interfaces

- **MobileVB** "ULSyncResult class" [*UltraLite for MobileVB User's Guide,* page 145]
- **ActiveX** "ULSyncResult class" [*UltraLite ActiveX User's Guide,* page 138]
- **UltraLite.NET** See "PartialDownloadRetained property" [*UltraLite.NET User's Guide,* page 286] (iAnywhere.Data.UltraLite namespace) and "PartialDownloadRetained property" [*UltraLite.NET User's Guide,* page 544] (iAnywhere.UltraLite namespace).
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncResult** in the API Reference.
- **C++ Component, embedded SQL, and Static C++ API** See "partial_download_retained synchronization parameter" [*UltraLite C/C++ User's Guide,* page 429].
- **Static Java API** Not available.
- **UltraLite for M-Business Anywhere** See "Class SyncResult" [*UltraLite for M-Business Anywhere User's Guide,* page 120].

See also

- "Resuming failed downloads" [*MobiLink Administration Guide,* page 74]
- "Keep Partial Download synchronization parameter" on page 321
- "Resume Partial Download synchronization parameter" on page 327

Password synchronization parameter

Function	A string specifying the MobiLink password associated with the user name. This user name and password are separate from any database user ID and password, and serves to identify and authenticate the application to the MobiLink synchronization server.
Default	The parameter is optional, and is a string.

See also

"Authenticating MobiLink Users" on page 9.

Interfaces

- **MobileVB** "ULSyncParms class" [*UltraLite for MobileVB User's Guide,* page 141]
- **ActiveX** "ULSyncParms class" [*UltraLite ActiveX User's Guide,* page 134]
- **UltraLite.NET** See "Password property" [*UltraLite.NET User's Guide,* page 268] (iAnywhere.Data.UltraLite namespace) and "Password property" [*UltraLite.NET User's Guide,* page 496] (iAnywhere.UltraLite namespace).
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.
- **C++ Component, embedded SQL, and Static C++ API** See "password synchronization parameter" [*UltraLite C/C++ User's Guide,* page 430].
- **Static Java API** See "password synchronization parameter" [*UltraLite Static Java User's Guide,* page 76].
- **UltraLite for M-Business Anywhere** See "Class SyncParms" [*UltraLite for M-Business Anywhere User's Guide,* page 112].

Ping synchronization parameter

Function

Confirm communications between the UltraLite client and the MobiLink synchronization server. When this parameter is set to true, no synchronization takes place.

When the MobiLink synchronization server receives a ping request, it connects to the consolidated database, authenticates the user, and then sends the authenticating user status and value back to the client.

If the ping succeeds, the MobiLink server issues an information message. If the ping does not succeed, it issues an error message.

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 ml_user.

The MobiLink synchronization server may execute the following scripts, if they exist, for a ping request:

- begin_connection
- authenticate_user
- authenticate_user_hashed

- end_connection

Default: The parameter is optional, and is a boolean.

See also: “-pi option” on page 144

Interfaces:

- **MobileVB** “ULSyncParms class” [*UltraLite for MobileVB User’s Guide,* page 141]
- **ActiveX** “ULSyncParms class” [*UltraLite ActiveX User’s Guide,* page 134]
- **UltraLite.NET** See “PingOnly property” [*UltraLite.NET User’s Guide,* page 268] (iAnywhere.Data.UltraLite namespace) and “PingOnly property” [*UltraLite.NET User’s Guide,* page 496] (iAnywhere.UltraLite namespace).
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.
- **C++ Component, embedded SQL, and Static C++ API** See “ping synchronization parameter” [*UltraLite C/C++ User’s Guide,* page 431].
- **Static Java API** See “ping synchronization parameter” [*UltraLite Static Java User’s Guide,* page 77].
- **UltraLite for M-Business Anywhere** See “Class SyncParms” [*UltraLite for M-Business Anywhere User’s Guide,* page 112].

Publication synchronization parameter

Function: Specifies the publications to be synchronized.

Default: If you do not specify a publication, all data is synchronized.

Usage: When synchronizing, set the publication parameter to a **publication mask**: an OR’d list of publication constants.

See also: “The UltraLite Generator” [*UltraLite Database User’s Guide,* page 89]

“Designing sets of data to synchronize separately” on page 280

Interfaces:

- **MobileVB** “ULPublicationSchema class” [*UltraLite for MobileVB User’s Guide,* page 121]
- **ActiveX** “ULPublicationSchema class” [*UltraLite ActiveX User’s Guide,* page 116]
- **UltraLite.NET** See “ULPublicationSchema class” [*UltraLite.NET User’s Guide,* page 226] (iAnywhere.Data.UltraLite namespace) and “PublicationSchema class” [*UltraLite.NET User’s Guide,* page 448] (iAnywhere.UltraLite namespace).

- **Native UltraLite for Java** See **ianywhere.native_ultralite.PublicationSchema** in the API Reference.
- **C++ Component, embedded SQL, and Static C++ API** See "publication synchronization parameter" [*UltraLite C/C++ User's Guide,* page 432].
- **Static Java API** See "publication synchronization parameter" [*UltraLite Static Java User's Guide,* page 78].
- **UltraLite for M-Business Anywhere** See "Class PublicationSchema" [*UltraLite for M-Business Anywhere User's Guide,* page 95].

Resume Partial Download synchronization parameter

Function

Resume a failed download. The synchronization does not upload changes, and downloads only those changes that were to be downloaded in the failed download.

Default

False.

See also

- "Resuming failed downloads" [*MobiLink Administration Guide,* page 74]
- "Keep Partial Download synchronization parameter" on page 321
- "Partial Download Retained synchronization parameter" on page 324

Interfaces

- **MobileVB** "ULSyncParms class" [*UltraLite for MobileVB User's Guide,* page 141]
- **ActiveX** "ULSyncParms class" [*UltraLite ActiveX User's Guide,* page 134]
- **UltraLite.NET** See "ResumePartialDownload property" [*UltraLite.NET User's Guide,* page 269] (iAnywhere.Data.UltraLite namespace) and "ResumePartialDownload property" [*UltraLite.NET User's Guide,* page 497] (iAnywhere.UltraLite namespace).
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.
- **C++ Component, embedded SQL, and Static C++ API** See "resume_partial_download synchronization parameter" [*UltraLite C/C++ User's Guide,* page 433].
- **Static Java API** Not available.
- **UltraLite for M-Business Anywhere** See "Class SyncParms" [*UltraLite for M-Business Anywhere User's Guide,* page 112].

Security synchronization parameter

Function

Set the UltraLite client to use Certicom encryption technology when exchanging messages with the MobiLink synchronization server.

> **Separately licensable option required**
> Transport-layer security requires that you obtain the separately-licensable SQL Anywhere Studio security option and is subject to export regulations.
>
> ☞ To order this component, see "Separately-licensable components" [*Introducing SQL Anywhere Studio*, page 5].

This parameter is not used in the static Java API. To use secure synchronization from the static Java API, choose a separate stream. For more information, see "UlSecureRSASocketStream synchronization parameters" on page 349 and "UlSecureSocketStream synchronization parameters" on page 350.

Default

The parameter is null by default, corresponding to no transport-layer security.

Usage

The security stream is specified in addition to the synchronization stream. Allowed values are as follows:

- **ULSecureCerticomTLSStream()** Elliptic-curve transport-layer security provided by Certicom. If you use this stream, you must link your application against *ulecc.llib* or supply *ulecc9.dll* with your application (*ulecc9w.dll* for Unicode applications).

- **ULSecureRSATLSStream()** RSA transport-layer security provided by Certicom. If you use this stream, you must link your application against *ulrsa.lib* or supply *ulrsa9.dll* with your application (*ulrsa9w.dll* for Unicode applications).

Example

The following C++ code sets synchronization to use RSA transport-layer security over TCP/IP:

```
//C++
ul_synch_info info;
...
info.stream = ULSocketStream();
info.security = ULSecureRSATLSStream();
```

See also

- "MobiLink Transport-Layer Security" [*MobiLink Administration Guide*, page 165]
- "Security Parameters synchronization parameter" on page 329
- "certificate_company" on page 37
- "certificate_name" on page 38

- “certificate_unit” on page 40
- “trusted_certificates” on page 53“certificate_company” on page 37

Interfaces

- **C++ Component, embedded SQL, and Static C++ API** See “security synchronization parameter” [*UltraLite C/C++ User’s Guide*, page 434].
- **Static Java API** Use a separate synchronization stream. See “UlSecureRSASocketStream synchronization parameters” on page 349, and “UlSecureSocketStream synchronization parameters” on page 350.

Security Parameters synchronization parameter

Function

Sets the options required when using transport-layer security. This parameter must be used together with the **security** parameter.

☞ For more information, see “Security synchronization parameter” on page 328.

This parameter is not applicable to static Java applications. To use secure synchronization from UltraLite static Java applications, choose a separate stream. For more information, see “UlSecureRSASocketStream synchronization parameters” on page 349 and “UlSecureSocketStream synchronization parameters” on page 350.

Default

The default setting is an empty string.

Usage

The ULSecureCerticomTLSStream() and ULSecureRSATLSStream() security parameters take a string composed of the following options, supplied in an semicolon-separated string.

- **certificate_company** The UltraLite application only accepts server certificates when the organization field on the certificate matches this value. By default, this field is not checked.
- **certificate_unit** The UltraLite application only accepts server certificates when the organization unit field on the certificate matches this value. By default, this field is not checked.
- **certificate_name** The UltraLite application only accepts server certificates when the common name field on the certificate matches this value. By default, this field is not checked.
- **trusted_certificates** If this value is supplied, the UltraLite application retrieves the certificat from permanent storage rather than from the database schema itself. The option cannot be used on Palm OS.

For example:

```
ul_synch_info info;
...
info.stream = ULSocketStream();
info.security = ULSecureCerticomTLSStream();
info.security_parms =
     UL_TEXT( "certificate_company=Sybase" )
     UL_TEXT( ";" )
     UL_TEXT( "certificate_unit=Sales" );
```

If you use secure synchronization on the Palm OS, trusted_certificates is not available. You must use the `-r` command-line option on the UltraLite generator to embed the certificate in the application itself. For more information, see "The UltraLite Generator" [*UltraLite Database User's Guide*, page 89].

See also

- "Security synchronization parameter" on page 328
- "certificate_company" on page 37
- "certificate_name" on page 38
- "certificate_unit" on page 40
- "trusted_certificates" on page 53"certificate_company" on page 37

Interfaces

- **C++ Component, embedded SQL, and Static C++ API** See "security_parms synchronization parameter" [*UltraLite C/C++ User's Guide*, page 435].
- **Static Java API** Use a separate synchronization stream.

Send Column Names synchronization parameter

Function

When set to true, UltraLite sends each column name to the MobiLink synchronization server.

This parameter is typically used together with the `-za` or `-ze` switch on the MobiLink synchronization server for automatically generating synchronization scripts.

This parameter is not available for UltraLite static Java applications.

Default

False.

See also

"-za option" [*MobiLink Administration Guide*, page 219]

Interfaces

- **MobileVB** "ULSyncParms class" [*UltraLite for MobileVB User's Guide*, page 141]
- **ActiveX** "ULSyncParms class" [*UltraLite ActiveX User's Guide*, page 134]
- **UltraLite.NET** See "SendColumnNames property" [*UltraLite.NET User's Guide*, page 270] (iAnywhere.Data.UltraLite namespace) and

"SendColumnNames property" [*UltraLite.NET User's Guide,* page 497] (iAnywhere.UltraLite namespace).

- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.
- **C++ Component, embedded SQL, and Static C++ API** See "send_column_names synchronization parameter" [*UltraLite C/C++ User's Guide,* page 436].
- **Static Java API** This feature is not available in the static Java API.
- **UltraLite for M-Business Anywhere** See "Class SyncParms" [*UltraLite for M-Business Anywhere User's Guide,* page 112].

Send Download Acknowledgement synchronization parameter

Function

Set this boolean parameter to true to instruct the MobiLink synchronization server that the client will provide a download acknowledgement.

If the client does send a download acknowledgement, the MobiLink synchronization server worker thread must wait for the client to apply the download. If the client does not sent a download acknowledgement, the MobiLink synchronization server is freed up sooner for its next synchronization.

Default

False.

Interfaces

- **MobileVB** "ULSyncParms class" [*UltraLite for MobileVB User's Guide,* page 141]
- **ActiveX** "ULSyncParms class" [*UltraLite ActiveX User's Guide,* page 134]
- **UltraLite.NET** See "SendDownloadAck property" [*UltraLite.NET User's Guide,* page 270] (iAnywhere.Data.UltraLite namespace) and "SendDownloadAck property" [*UltraLite.NET User's Guide,* page 498] (iAnywhere.UltraLite namespace).
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.
- **C++ Component, embedded SQL, and Static C++ API** See "send_download_ack synchronization parameter" [*UltraLite C/C++ User's Guide,* page 437].
- **Static Java API** This parameter is not available for static Java applications.

- **UltraLite for M-Business Anywhere** See "Class SyncParms" [*UltraLite for M-Business Anywhere User's Guide*, page 112].

Stream Error synchronization parameter

Function	Set a structure to hold communications error reporting information.
Applies To	This parameter applies only to C/C++ interfaces.

UltraLite components other than the UltraLite C++ Component receive communications error information as part of the Sync Result parameter. See "Sync Result synchronization parameter" on page 335.

This feature is not available for UltraLite static Java applications.

Default

The parameter is set by the MobiLink synchronization server, and so is read-only. It is set only if a communication error occurs during synchronization.

Interfaces

- **MobileVB** "ULSyncResult class" [*UltraLite for MobileVB User's Guide*, page 145]
- **ActiveX** "ULSyncResult class" [*UltraLite ActiveX User's Guide*, page 138]
- **UltraLite.NET** See ianywhere.UltraLite.SyncResult.
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.
- **C++ Component, embedded SQL, and Static C++ API** See "stream_error synchronization parameter" [*UltraLite C/C++ User's Guide*, page 440].
- **Static Java API** This feature is not available in static Java.
- **UltraLite for M-Business Anywhere** See "Class SyncResult" [*UltraLite for M-Business Anywhere User's Guide*, page 120].

Stream Type synchronization parameter

Function

Set the MobiLink network protocol to use for synchronization.

Most network protocols require protocol options to identify the MobiLink synchronization server address and other behavior. These options are supplied in the **stream_parms** parameter.

☞ For more information, see "Stream Parameters synchronization parameter" on page 334.

Default

This parameter is required. It has no default value.

Usage

When the network protocol requires an option, pass that option using the Stream Parameters parameter; otherwise, set the Stream Parameters parameter to null.

The following stream functions are available, but not all are available on all target platforms:

Network protocol	**Description**
ActiveSync	ActiveSync synchronization (Windows CE only). ☞ For a list of protocol options, see "ActiveSync protocol options" on page 341.
HTTP	Synchronize via HTTP. The HTTP protocol uses TCP/IP as its underlying transport. UltraLite applications act as Web browsers and the MobiLink synchronization server acts as a Web server. UltraLite applications send POST requests to send data to the server and GET requests to read data from the server. ☞ For a list of protocol options, see "HTTP protocol options" on page 346.
HTTPS and HTTPS_-FIPS	Synchronize via the HTTPS protocol. The HTTPS protocol uses SSL or TLS as its underlying protocol. It operates over Internet protocols (HTTP and TCP/IP). The HTTPS protocol requires the use of technology supplied by Certicom. Use of Certicom technology requires that you obtain the separately-licensable SQL Anywhere Studio security option and is subject to export regulations. For more information, see "Separately-licensable components" [*Introducing SQL Anywhere Studio*, page 5]. ☞ For a list of protocol options, see "HTTPS protocol options" on page 347.
TCP/IP	Synchronize via TCP/IP. ☞ For a list of protocol options, see "TCP/IP protocol options" on page 345.

Network protocol	Description
UlSecureSocketStream()	TCP/IP or HTTP synchronization with transport-layer security using elliptic curve encryption. This stream is available for static Java applications only. ☞ For a list of protocol options, see "UlSecureSocketStream synchronization parameters" on page 350.
UlSecureRSASocket-Stream()	TCP/IP or HTTP synchronization with transport-layer security using RSA encryption. This stream is available for static Java applications only. ☞ For a list of protocol options, see "UlSecureSocketStream synchronization parameters" on page 350.

Interfaces

- **MobileVB** "ULSyncParms class" [*UltraLite for MobileVB User's Guide*, page 141]
- **ActiveX** "ULSyncParms class" [*UltraLite ActiveX User's Guide*, page 134]
- **UltraLite.NET** See "Stream property" [*UltraLite.NET User's Guide*, page 270] (iAnywhere.Data.UltraLite namespace) and "Stream property" [*UltraLite.NET User's Guide*, page 498] (iAnywhere.UltraLite namespace).
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.
- **C++ Component, embedded SQL, and Static C++ API** See "stream synchronization parameter" [*UltraLite C/C++ User's Guide*, page 438].
- **Static Java API** See "stream synchronization parameter" [*UltraLite Static Java User's Guide*, page 81].
- **UltraLite for M-Business Anywhere** See "Class SyncParms" [*UltraLite for M-Business Anywhere User's Guide*, page 112].

Stream Parameters synchronization parameter

Function

Sets options to configure the network protocol.

A semi-colon separated list of network protocol options. Each option is of the form *keyword*=*value*, where the allowed sets of keywords depends on the network protocol.

For a list of available options for each protocol, see the following sections:

- ♦ “ActiveSync protocol options” on page 341
- ♦ “HotSync protocol options” on page 343
- ♦ “HTTP protocol options” on page 346
- ♦ “HTTPS protocol options” on page 347
- ♦ “TCP/IP protocol options” on page 345
- ♦ “UlSecureRSASocketStream synchronization parameters” on page 349
- ♦ “UlSecureSocketStream synchronization parameters” on page 350

Default The parameter is optional, is a string, and by default is null.

See also “Network protocol options for UltraLite synchronization clients” on page 341.

Interfaces

- ♦ **MobileVB** “ULSyncParms class” [*UltraLite for MobileVB User’s Guide,* page 141]
- ♦ **ActiveX** “ULSyncParms class” [*UltraLite ActiveX User’s Guide,* page 134]
- ♦ **UltraLite.NET** See “StreamParms property” [*UltraLite.NET User’s Guide,* page 271] (iAnywhere.Data.UltraLite namespace) and “StreamParms property” [*UltraLite.NET User’s Guide,* page 498] (iAnywhere.UltraLite namespace).
- ♦ **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.
- ♦ **C++ Component, embedded SQL, and Static C++ API** See “stream_parms synchronization parameter” [*UltraLite C/C++ User’s Guide,* page 443].
- ♦ **Static Java API** See “stream_parms synchronization parameter” [*UltraLite Static Java User’s Guide,* page 83].
- ♦ **UltraLite for M-Business Anywhere** See “Class SyncParms” [*UltraLite for M-Business Anywhere User’s Guide,* page 112].

Sync Result synchronization parameter

Function Reports the status of a synchronization.

Applies To This parameter applies only to the UltraLite components.

The C/C++ interfaces receive this information in separate parameters.

Communications error information is not available for UltraLite static Java applications.

Default — The parameter is set by the MobiLink synchronization server, and so is read-only. It is set only if a communication error occurs during synchronization.

Remarks — Sync Result is a compound parameter containing a variety of information in separate fields:

- **Authentication Status** Reports success or failure of authentication. See “Authentication Status synchronization parameter” on page 317.
- **Ignored Rows** Reports the number of ignored rows. See “Ignored Rows synchronization parameter” on page 320.
- **Stream Error information** The Stream Error information includes a Stream Error Code, Stream Error Context, Stream Error ID, and Stream Error System.
- **Upload OK** Reports the success or failure of the upload phase. See “Upload OK synchronization parameter” on page 336.

Interfaces

- **MobileVB** “ULSyncResult class” [*UltraLite for MobileVB User’s Guide*, page 145]
- **ActiveX** “ULSyncResult class” [*UltraLite ActiveX User’s Guide*, page 138]
- **UltraLite.NET** See “ULSyncResult class” [*UltraLite.NET User’s Guide*, page 284] (iAnywhere.Data.UltraLite namespace) and “SyncResult class” [*UltraLite.NET User’s Guide*, page 542] (iAnywhere.UltraLite namespace).
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.
- **C++ Component, embedded SQL,, Static C++ API** See “stream_error synchronization parameter” [*UltraLite C/C++ User’s Guide*, page 440].
- **Static Java API** This feature is not available in static Java.
- **UltraLite for M-Business Anywhere** See “Class SyncResult” [*UltraLite for M-Business Anywhere User’s Guide*, page 120].

Upload OK synchronization parameter

Function — Reports the status of data uploaded to the MobiLink synchronization server.

Usage — The MobiLink synchronization server sets this parameter, and so it is read-only.

After synchronization, the parameter holds **true** if the upload was successful, and **false** otherwise. You can check this parameter if there was a synchronization error, to know whether data was successfully uploaded before the error occurred.

Interfaces

- **MobileVB** "ULSyncResult class" [*UltraLite for MobileVB User's Guide*, page 145]
- **ActiveX** "ULSyncResult class" [*UltraLite ActiveX User's Guide*, page 138]
- **UltraLite.NET** See "UploadOK property" [*UltraLite.NET User's Guide*, page 287] (iAnywhere.Data.UltraLite namespace) and "UploadOK property" [*UltraLite.NET User's Guide*, page 545] (iAnywhere.UltraLite namespace).
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.
- **C++ Component, embedded SQL, and Static C++ API** See "upload_ok synchronization parameter" [*UltraLite C/C++ User's Guide*, page 444].
- **Static Java API** See "upload_ok synchronization parameter" [*UltraLite Static Java User's Guide*, page 84].
- **UltraLite for M-Business Anywhere** See "Class SyncResult" [*UltraLite for M-Business Anywhere User's Guide*, page 120].

Upload Only synchronization parameter

Function

Indicates that there should be no downloads in the current synchronization, which can save communication time, especially over slow communication links. When set to true, the client waits for the upload acknowledgement from the MobiLink synchronization server, after which it terminates the synchronization session successfully.

Default

The parameter is an optional Boolean value, and by default is false.

See also

"Synchronizing high-priority changes" on page 282

"Download Only synchronization parameter" on page 320

Interfaces

- **MobileVB** "ULSyncParms class" [*UltraLite for MobileVB User's Guide*, page 141]
- **ActiveX** "ULSyncParms class" [*UltraLite ActiveX User's Guide*, page 134]
- **UltraLite.NET** See "UploadOnly property" [*UltraLite.NET User's Guide*, page 271] (iAnywhere.Data.UltraLite namespace) and "UploadOnly

property" [*UltraLite.NET User's Guide,* page 499] (iAnywhere.UltraLite namespace).

- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.
- **C++ Component, embedded SQL, and Static C++ API** See "upload_only synchronization parameter" [*UltraLite C/C++ User's Guide,* page 445].
- **Static Java API** See "upload_only synchronization parameter" [*UltraLite Static Java User's Guide,* page 85].
- **UltraLite for M-Business Anywhere** See "Class SyncParms" [*UltraLite for M-Business Anywhere User's Guide,* page 112].

User Data synchronization parameter

Function

Make application-specific information available to the synchronization observer.

Usage

When implementing the synchronization observer callback function or event handler, you can make application-specific information available by providing information using the User Data parameter.

Some components, such as UltraLite.NET, do not require a separate parameter to handle user data and so have no User Data parameter.

See also

"Observer synchronization parameter" on page 323

Interfaces

- **MobileVB** "ULConnection class" [*UltraLite for MobileVB User's Guide,* page 91]
- **ActiveX** "ULDatabaseManager class" [*UltraLite ActiveX User's Guide,* page 101]
- **Native UltraLite for Java** See **ianywhere.native_ultralite.Connection** in the API Reference.
- **C++ Component, embedded SQL, and Static C++ API** See "user_data synchronization parameter" [*UltraLite C/C++ User's Guide,* page 446].
- **Static Java API** See "user_data synchronization parameter" [*UltraLite Static Java User's Guide,* page 86].

User Name synchronization parameter

Function

A string specifying the user name that uniquely identifies the MobiLink client to the MobiLink synchronization server. MobiLink uses this value to determine the download content, to record the synchronization state, and to recover from interruptions during synchronization.

Default — The parameter has no default value, and must be explicitly set.

Usage — The user name is required unless the MobiLink synchronization server is being run with user authentication turned off. For more information, see "-zu option" [*MobiLink Administration Guide,* page 222].

See also — "Authenticating MobiLink Users" on page 9.

"MobiLink users" on page 7.

Interfaces

- **MobileVB** "ULSyncParms class" [*UltraLite for MobileVB User's Guide,* page 141]
- **ActiveX** "ULSyncParms class" [*UltraLite ActiveX User's Guide,* page 134]
- **UltraLite.NET** See "UserName property" [*UltraLite.NET User's Guide,* page 272] (iAnywhere.Data.UltraLite namespace) and "UserName property" [*UltraLite.NET User's Guide,* page 499] (iAnywhere.UltraLite namespace).
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.
- **C++ Component, embedded SQL, and Static C++ API** See "user_name synchronization parameter" [*UltraLite C/C++ User's Guide,* page 447].
- **Static Java API** See "user_name synchronization parameter" [*UltraLite Static Java User's Guide,* page 87].
- **UltraLite for M-Business Anywhere** See "Class SyncParms" [*UltraLite for M-Business Anywhere User's Guide,* page 112].

Version synchronization parameter

Function — Each synchronization script in the consolidated database is marked with a version string. For example, there may be two different **download_cursor** scripts, identified by different version strings. The version string allows an UltraLite application to choose from a set of synchronization scripts.

Default — This parameter is required and is a string.

See also — "Script versions" [*MobiLink Administration Guide,* page 239].

Interfaces

- **MobileVB** "ULSyncParms class" [*UltraLite for MobileVB User's Guide,* page 141]
- **ActiveX** "ULSyncParms class" [*UltraLite ActiveX User's Guide,* page 134]

- **UltraLite.NET** See "Version property" [*UltraLite.NET User's Guide*, page 272] (iAnywhere.Data.UltraLite namespace) and "Version property" [*UltraLite.NET User's Guide*, page 500] (iAnywhere.UltraLite namespace).
- **Native UltraLite for Java** See **ianywhere.native_ultralite.SyncParms** in the API Reference.
- **C++ Component, embedded SQL, and Static C++ API** See "version synchronization parameter" [*UltraLite C/C++ User's Guide*, page 448].
- **Static Java API** See "version synchronization parameter" [*UltraLite Static Java User's Guide*, page 88].
- **UltraLite for M-Business Anywhere** See "Class SyncParms" [*UltraLite for M-Business Anywhere User's Guide*, page 112].

Network protocol options for UltraLite synchronization clients

This section lists the options you can use with each network protocol. The network protocol options provide information such as addressing information (host and port) and protocol-specific information to ensure that the client can locate and properly communicate with the MobiLink synchronization server.

ActiveSync protocol options

The ActiveSync synchronization stream is accessible only from Native UltraLite for Java, embedded SQL, and static C++ API applications running on Windows CE.

To choose ActiveSync synchronization:

♦ In Native UltraLite for Java, supply StreamType.ACTIVE_SYNC as the argument to the syncParms.setStream method. For example:

```
_conn.syncParms.setStream( StreamType.ACTIVE_SYNC );
```

☞ For more information, see **ianywhere.native_ultralite.StreamType** and **ianywhere.native_ultralite.SyncParms** in the Native UltraLite for Java API Reference.

♦ In embedded SQL and the static C++ API, supply ULActiveSyncStream() as the network protocol. For example:

```
ul_synch_info info;
...
info.stream = ULActiveSyncStream();
```

☞ For more information, see "ULActiveSyncStream function" [*UltraLite C/C++ User's Guide*, page 363].

Meaning of protocol options

The protocol options control the connection from the MobiLink ActiveSync provider, running on the desktop machine, to the MobiLink synchronization server.

The protocol options take the following form:

```
stream=stream_name;provider_stream_parameters
```

where *stream_name* indicates the protocol for the conduit to use when communicating from the conduit to the MobiLink synchronization server. It must be one of the following:

♦ **tcpip**

- **http**
- **https**

and where *provider_stream_parameters* is a set of protocol options for use by the ActiveSync provider, and has the same form as the protocol options for the protocol in use. For the given protocol, the *provider_stream_parameters* adopts the same defaults as the protocol options for the protocol. The default value for the *stream_name* is **tcpip**.

For example, the following static C++ code uses an HTTP protocol:

```
ULInitSynchInfo( &info );
info.stream = ULActiveSyncStream();
info.stream_parms = "stream=http";
ULSynchronize( &sqlca, &info );
```

☞ For more information on *provider_stream_parameters*, see "TCP/IP protocol options" on page 345, "HTTP protocol options" on page 346, and "HTTPS protocol options" on page 347.

Adding encryption to ActiveSync synchronization

To add Certicom encryption to the stream, the root certificates must be in a file on the desktop machine. This is different from other UltraLite applications, where the encryption information is embedded in the **security** synchronization parameter.

The protocol options need to be specified in much the same way as for Adaptive Server Anywhere MobiLink clients . The format is:

security=*cipher*{ *keyword***=***value***;**. . . }

where *cipher* must be certicom_tls and the keywords are taken from the following list:

- **certificate_company** The organization field on the certificate.
- **certificate_unit** The organization unit field on the certificate.
- **certificate_name** The common name field on the certificate.
- **trusted_certificates** The location of the trusted certificates.

For example, a static C++ application may use a line such as the following:

```
info.stream_parms = "stream=tcpip;security=ecc_tls(trusted_
         certificates=trusted.crt)";
```

☞ For more information, see:

- "certificate_company" on page 37
- "certificate_name" on page 38

- "certificate_unit" on page 40
- "trusted_certificates" on page 53

HotSync protocol options

The HotSync synchronization stream is accessible only from UltraLite for MobileVB applications, embedded SQL applications, and static C++ API applications running on the Palm Computing Platform. Unlike HTTP or TCP/IP synchronization, HotSync synchronization is initiated externally by the HotSync Manager, rather than by a synchronization function within the UltraLite application.

To choose HotSync synchronization:

- In UltraLite for MobileVB, choose ulPalmConduit from the ULStreamType enumeration as the ULSyncParms.Stream.

  ☞ For more information, see "ULSyncParms class" [*UltraLite for MobileVB User's Guide*, page 141].

- In embedded SQL or the static C++ API, supply the ul_synch_info structure to the ULSetSynchInfo method of your application. The stream parameter is ignored and may be set to UL_NULL.

  ☞ For more information, see "ULSetSynchInfo function" [*UltraLite C/C++ User's Guide*, page 386].

Meaning of protocol options

For HotSync synchronization, the protocol options do *not* control the connection from the device to the HotSync Manager or HotSync Server. Instead, they specify the connection from the MobiLink conduit, running at the HotSync manager or server, to the MobiLink synchronization server.

The argument has the following form:

```
stream=stream_name;conduit_stream_parameters
```

where *stream_name* indicates the protocol for the conduit to use when communicating from the conduit to the MobiLink synchronization server. It must be one of the following:

- **tcpip**
- **http**
- **https**

and where *conduit_stream_parameters* is a set of protocol options for use by the conduit, and has the same form as the **stream_parms** argument for the protocol in use. For the given stream, the *conduit_stream_parameters* adopts the same defaults as the **stream_parms** argument for the protocol. The default value for the *stream_name* is tcpip.

For example, the following embedded SQL code uses an HTTP synchronization stream:

```
ULInitSynchInfo( &info );
info.stream_parms = "stream=http";
```

☞ For more information on *conduit_stream_parameters*, see "TCP/IP protocol options" on page 345, "HTTP protocol options" on page 346, and "HTTPS protocol options" on page 347.

Null value and default settings

If you use HotSync synchronization, and do not supply protocol options, the conduit searches in the registry for the protocol name and protocol options. If it finds no valid network protocol, the default protocol and protocol options are used. This default stream parameter setting is:

```
stream=tcpip;host=localhost
```

Adding encryption to HotSync synchronization

To add Certicom encryption to the stream, the root certificates must be in a file on the desktop machine. This is different from other UltraLite applications, where the encryption information is embedded in the **security** synchronization parameter.

The protocol options need to be specified in much the same way as for Adaptive Server Anywhere MobiLink clients . The format is:

security=*cipher*{ *keyword***=***value***;**. . . }

where *cipher* must be certicom_tls and the keywords are taken from the following list:

- ♦ **certificate_company** The organization field on the certificate.
- ♦ **certificate_unit** The organization unit field on the certificate.
- ♦ **certificate_name** The common name field on the certificate.
- ♦ **trusted_certificates** The location of the trusted certificates.

For example, in a static C++ application:

```
info.stream_parms = "stream=tcpip;security=ecc_tls(trusted_
         certificates=trusted.crt)";
```

☞ For more information, see:

- ♦ "certificate_company" on page 37
- ♦ "certificate_name" on page 38
- ♦ "certificate_unit" on page 40

- "trusted_certificates" on page 53

TCP/IP protocol options

The TCP/IP synchronization stream is accessible from all UltraLite interfaces.

Selecting the TCP/IP synchronization stream

To select TCP/IP as the synchronization stream:

- In UltraLite for MobileVB and UltraLite ActiveX, choose ulTCPIP from the ULStreamType enumeration as the ULSyncParms.Stream.

  ☞ For more information, see "ULSyncParms class" [*UltraLite for MobileVB User's Guide*, page 141] and "ULSyncParms class" [*UltraLite ActiveX User's Guide*, page 134].

- In Native UltraLite for Java, supply StreamType.TCPIP as the argument for SyncParms.setStream().

  ☞ For more information, see **ianywhere.native_ultralite.StreamType** and **ianywhere.native_ultralite.SyncParms** in the Native UltraLite for Java API Reference.

- In embedded SQL or the static C++ API, supply ULSocketStream() as the stream synchronization parameter.

  ☞ For more information, see "ULSocketStream function" [*UltraLite C/C++ User's Guide*, page 387].

- In the static Java API, supply UlSocketStream as the argument for UlSynchOptions.setStream(). For example:

  ```
  UlSynchOptions opts = new UlSynchOptions;
  opts.setStream(new UlSocketStream() );
  ```

  For more information, see "stream synchronization parameter" [*UltraLite Static Java User's Guide*, page 81].

Protocol options

When you use the TCP/IP protocol, you can choose from the following protocol options:

Parameter	For more information, see...
client_port= *nnnnn*[-*mmmmm*]	"client_port" on page 41
host=*hostname*	"host" on page 42
liveness_timeout=*n*	"liveness_timeout" on page 46
network_connect_-timeout=*seconds*	"network_connect_timeout" on page 47
network_leave_-open={**0**|**1**}	"network_leave_open" on page 47
network_-name=*name*	"network_name" on page 48
port=*portnumber*	"port" on page 49

HTTP protocol options

The HTTP synchronization stream is accessible from all UltraLite components.

Selecting the HTTP synchronization stream

To select HTTP as the synchronization stream:

♦ In UltraLite for MobileVB and UltraLite for ActiveX, choose ulHTTP from the ULStreamType enumeration as the ULSyncParms.Stream.

☞ For more information, see "ULSyncParms class" [*UltraLite for MobileVB User's Guide,* page 141].

♦ In Native UltraLite for Java, supply StreamType.HTTP as the argument for SyncParms.setStream().

☞ For more information, see **ianywhere.native_ultralite.StreamType** and **ianywhere.native_ultralite.SyncParms** in the Native UltraLite for Java API Reference.

♦ In embedded SQL or the static C++ API, supply ULHTTPStream() as the stream synchronization parameter.

☞ For more information, see "ULHTTPStream function" [*UltraLite C/C++ User's Guide,* page 375].

Protocol options

When you use the HTTP stream, you can choose from the following protocol options:

Parameter	For more information, see...
buffer_size=*nnnn*	"buffer_size" on page 36
client_port= *nnnnn*[-*mmmmm*]	"client_port" on page 41
custom_header=*header*	"custom_header" on page 41
host=*hostname*	"host" on page 42
network_connect_-timeout=*seconds*	"network_connect_timeout" on page 47
network_leave_-open={**0**|**1**}	"network_leave_open" on page 47
network_name=*name*	"network_name" on page 48
persistent={**0**|**1**}	"persistent" on page 48
port=*portnumber*	"port" on page 49
proxy_host=*proxy_-hostname*	"proxy_host" on page 50
proxy_port= *proxy_-portnumber*	"proxy_port" on page 50
set_cookie=*cookie_-name=cookie_value*	"set_cookie" on page 52
url_suffix=*suffix*	"url_suffix" on page 55
version=*versionnumber*	"version" on page 56

See also

"Configuring MobiLink clients and servers for the Redirector" [*MobiLink Administration Guide*, page 137]

HTTPS protocol options

The HTTPS synchronization stream is accessible from all UltraLite components.

Selecting the HTTPS synchronization stream

To select HTTPS as the synchronization stream:

♦ In UltraLite for MobileVB and UltraLite for eMbedded Visual Basic, choose ulHTTPS from the ULStreamType enumeration as the ULSyncParms.Stream.

☞ For more information, see "ULSyncParms class" [*UltraLite for MobileVB User's Guide*, page 141].

♦ In Native UltraLite for Java, supply StreamType.HTTPS as the argument for SyncParms.setStream().

  ☞ For more information, see **ianywhere.native_ultralite.StreamType** and **ianywhere.native_ultralite.SyncParms** in the Native UltraLite for Java API Reference.

♦ In embedded SQL or the static C++ API, supply ULHTTPSStream() as the stream synchronization parameter.

  ☞ For more information, see "ULHTTPSStream function" [*UltraLite C/C++ User's Guide*, page 374].

Separately licensable option required
Transport-layer security requires that you obtain the separately-licensable SQL Anywhere Studio security option and is subject to export regulations.

☞ To order this component, see "Separately-licensable components" [*Introducing SQL Anywhere Studio*, page 5].

Protocol options

When you use the HTTPS stream, you can choose from the following protocol options:

Parameter	For more information, see...
buffer_size=*nnnn*	"buffer_size" on page 36
certificate_-company=*company_name*	"certificate_company" on page 37
certificate_name=*name*	"certificate_name" on page 38
certificate_-unit=*company_unit*	"certificate_unit" on page 40
client_port= *nnnnn*[-*mmmmm*]	"client_port" on page 41
custom_header=*header*	"custom_header" on page 41
host=*hostname*	"host" on page 42
network_connect_-timeout=*seconds*	"network_connect_timeout" on page 47
network_leave_-open={**0**|**1**}	"network_leave_open" on page 47

Parameter	For more information, see...
network_name=*name*	"network_name" on page 48
persistent={**0**|**1**}	"persistent" on page 48
port=*portnumber*	"port" on page 49
proxy_host=*proxy_-hostname*	"proxy_host" on page 50
proxy_port=*proxy_-portnumber*	"proxy_port" on page 50
set_cookie=*cookie_-name=cookie_value*	"set_cookie" on page 52
trusted_-certificates=*filename*	"trusted_certificates" on page 53
url_suffix=*suffix*	"url_suffix" on page 55
version=*versionnumber*	"version" on page 56

See also

- "Configuring UltraLite clients to use transport-layer security" [*MobiLink Administration Guide*, page 183]
- "Configuring MobiLink clients and servers for the Redirector" [*MobiLink Administration Guide*, page 137]

UlSecureRSASocketStream synchronization parameters

Transport-layer security using Certicom RSA encryption is accessed from static Java applications as a separate stream, accessed using the UlSecureRSASocketStream object. This is different behavior from other UltraLite applications, where a separate parameter is supplied to the synchronization structure.

Separately licensable option required
Transport-layer security requires that you obtain the separately-licensable SQL Anywhere Studio security option and is subject to export regulations.

☞ To order this component, see "Separately-licensable components" [*Introducing SQL Anywhere Studio*, page 5].

Protocol options

The parameters for the UlSecureSocketStream are supplied in a semicolon-separated string. When you use UlSecureRSASocketStream, you can choose from the following protocol options:

Parameter	Description
certificate_company=*company_-name*	“certificate_company” on page 37
certificate_name=*name*	“certificate_name” on page 38
certificate_unit=*company_unit*	“certificate_unit” on page 40
client_port= *nnnnn*[-*mmmmm*]	“client_port” on page 41
host=*hostname*	“host” on page 42
liveness_timeout=*n*	“liveness_timeout” on page 46
port=*portnumber*	“port” on page 49

UlSecureSocketStream synchronization parameters

Transport-layer security using Certicom elliptic-curve encryption is accessed from static Java applications as a separate stream, accessed using the UlSecureSocketStream object.

Separately licensable option required
Transport-layer security requires that you obtain the separately-licensable SQL Anywhere Studio security option and is subject to export regulations.

☞ To order this component, see “Separately-licensable components” [*Introducing SQL Anywhere Studio*, page 5].

☞ For more information, see “stream synchronization parameter” [*UltraLite Static Java User’s Guide*, page 81], and “Using transport-layer security” [*UltraLite Static Java User’s Guide*, page 42].

Protocol options

The parameters for the UlSecureSocketStream are supplied in a semicolon-separated string. When you use UlSecureRSASocketStream, you can choose from the following protocol options:

Parameter	Description
certificate_company=*company_-name*	"certificate_company" on page 37
certificate_name=*name*	"certificate_name" on page 38
certificate_unit=*company_unit*	"certificate_unit" on page 40
client_port= *nnnnn*[-*mmmmm*]	"client_port" on page 41
host=*hostname*	"host" on page 42
liveness_timeout=*n*	"liveness_timeout" on page 46
port=*portnumber*	"port" on page 49

Index