SYBASE iAnywhere

MobiLink™
Getting Started

Contents

About this book

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

About the SQL Anywhere documentation

The complete SQL Anywhere documentation is available in four formats:

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

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

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

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

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

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

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

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

Documentation conventions

This section lists the conventions used in this documentation.

Operating systems

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

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

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

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

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

For the complete list of platforms supported by SQL Anywhere, see "Supported platforms" [*SQL Anywhere 12 - Introduction*].

Directory and file names

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

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

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

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

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

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

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

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

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

  For example, on Windows, the network database server is *dbsrv12.exe*. On Unix, it is *dbsrv12*.

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

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

  For more information about the default location of *install-dir*, see "SQLANY12 environment variable" [*SQL Anywhere Server - Database Administration*].

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

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

  For more information about the default location of *samples-dir*, see "SQLANYSAMP12 environment variable" [*SQL Anywhere Server - Database Administration*].

Command prompts and command shell syntax

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

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

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

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

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

  Where parentheses cause syntax problems, substitute curly braces:

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

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

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

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

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

  In many shells, the value of the key would be `my "secret" key`.
- **Environment variables** The documentation refers to setting environment variables. In Windows shells, environment variables are specified using the syntax *%ENVVAR%*. In Unix shells, environment variables are specified using the syntax *$ENVVAR* or *${ENVVAR}*.

Contacting the documentation team

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

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

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

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

Finding out more and requesting technical support

Newsgroups

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

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

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

The newsgroups include the following:

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

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

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

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

Developer Centers

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

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

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

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

Introducing MobiLink Technology

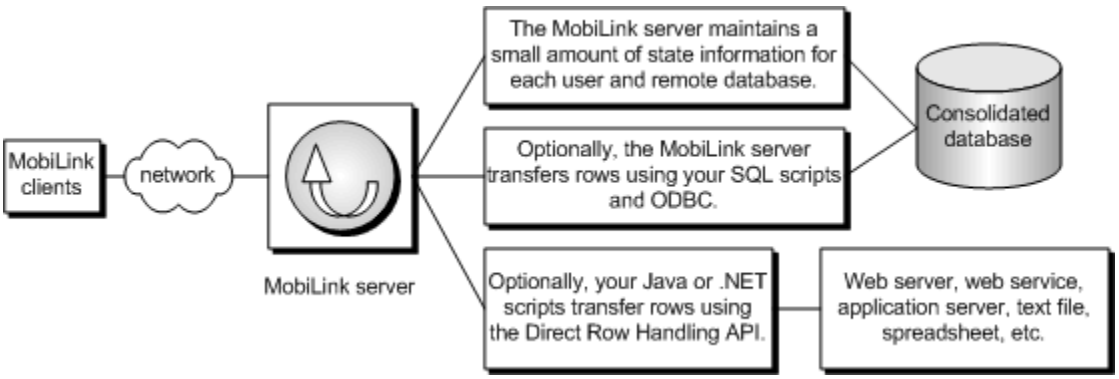
This section introduces MobiLink synchronization technology and describes how to use it to replicate data between two or more databases.

Understanding MobiLink synchronization

MobiLink is a session-based synchronization technology designed to synchronize UltraLite and SQL Anywhere remote databases with a consolidated database.

Parts of a MobiLink application

In MobiLink synchronization, many clients synchronize through the MobiLink server to central data sources.



- **MobiLink clients** The client can be installed on a handheld device, a server or desktop computer, or a smartphone. Two types of clients are supported: UltraLite and SQL Anywhere databases. Either or both can be used in a MobiLink installation. See "MobiLink clients" [*MobiLink - Client Administration*].

- **Network** The connection between the MobiLink server and the MobiLink client can use several protocols. See:

  - MobiLink server: "-x mlsrv12 option" [*MobiLink - Server Administration*]
  - UltraLite and SQL Anywhere clients: "MobiLink client network protocol options" [*MobiLink - Client Administration*]

- **MobiLink server** This server manages the synchronization process and provides the interface between all MobiLink clients and the consolidated database server. See "MobiLink server" [*MobiLink - Server Administration*].

- **Consolidated database** This database typically contains the central copy of your application information in the synchronization system. It also typically holds system tables and procedures that

are required by MobiLink synchronization, and state information needed to synchronize. See "MobiLink consolidated databases" [*MobiLink - Server Administration*].

- **State information** The MobiLink server typically maintains synchronization information in system tables in the consolidated database. It does this over an ODBC connection.

  You can also choose to store your state information in a separate database. See "MobiLink system database" [*MobiLink - Server Administration*].

- **SQL row handling** If you provide the MobiLink server with SQL scripts, it uses these scripts to transfer rows to and from the consolidated database over an ODBC connection. See "Options for writing server-side synchronization logic" on page 11.

- **Direct row handling** In addition to a consolidated database, you can optionally synchronize with other data sources using MobiLink direct row handling. See "Direct row handling" [*MobiLink - Server Administration*].

- **Synchronization scripts** You write synchronization scripts for each table in the remote database and you save these scripts in MobiLink system tables in the consolidated database. These scripts determine what is done with the uploaded data, and what data to download. There are two types of script: table scripts and connection-level scripts. See:

  - "Overview of MobiLink events" [*MobiLink - Server Administration*]
  - "Writing synchronization scripts" [*MobiLink - Server Administration*]
  - "Synchronization events" [*MobiLink - Server Administration*]
  - "Options for writing server-side synchronization logic" on page 11

MobiLink features

MobiLink synchronization is adaptable and flexible. The following are some of its key features:

Features

- **Easy to get started** Using the **Create Synchronization Model Wizard**, you can create synchronization applications quickly. The wizard can handle many difficult implementation details of complex synchronization systems. Sybase Central allows you to view a synchronization model offline, provides an easy interface for making changes, and has a deployment option for you to deploy the model to your consolidated database.

- **Monitoring and reporting** MobiLink provides three mechanisms for monitoring your synchronizations: the MobiLink Monitor, The SQL Anywhere Monitor for MobiLink, and statistical scripts.

- **Performance tuning** There are several mechanisms for tuning MobiLink performance. For example, you can adjust the degree of contention, upload cache size, number of database connections, logging verbosity, or BLOB cache size.

- **Scalability** MobiLink is an extremely scalable and robust synchronization platform. A single MobiLink server can handle thousands of simultaneous synchronizations, and multiple MobiLink

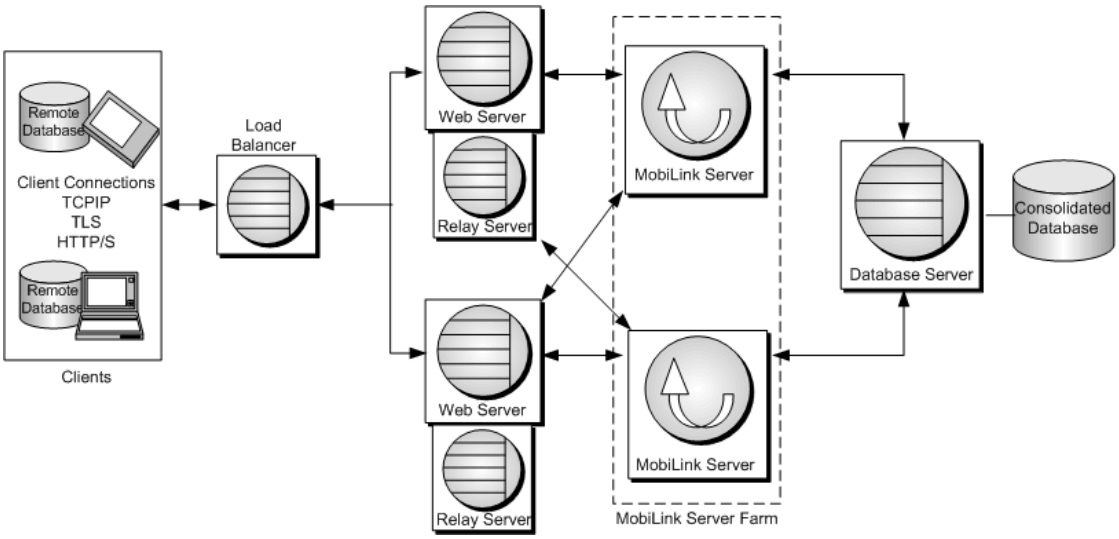
servers can be run simultaneously using load balancing. The MobiLink server is multi-threaded and uses connection pooling with the consolidated database.

- **Security** MobiLink provides extensive security options, including user authentication that can be integrated with your existing authentication, encryption, and transport-layer security that works by the exchange of secure certificates. MobiLink also provides FIPS-certified security options.

- **Relay server and Sybase Relay Server hosting service** The Relay Server enables secure, load-balanced communication between mobile devices and back-end servers through a web server. See "Introduction to the Relay Server" [*Relay Server*].

  The Sybase Relay Server hosting service is a farm of Relay Servers hosted by Sybase that enables you to more easily develop and evaluate mobile applications that use MobiLink data synchronization, especially where data is sent using public wireless networks. See "Sybase Hosted Relay Service" [*Relay Server*].

  The diagram below shows how the relay server fits into a MobiLink environment.



Architecture

- **Data coordination** MobiLink allows you to choose selected portions of the data for synchronization. MobiLink synchronization also allows you to resolve conflicts between changes made in different databases. The synchronization process is controlled by synchronization logic, which can be written as a SQL, Java, or .NET application. Each piece of logic is called a **script**. With scripts, for example, you can specify how uploaded data is applied to the consolidated database, specify what gets downloaded, and handle different schema and names between the consolidated and remote databases. Event-based scripting provides great flexibility in the design of the synchronization process, including such features as conflict resolution, error reporting, and user authentication.

- **Two-way synchronization** Changes to a database can be made at any location.

- **Upload-only or download-only synchronization** By default synchronization is two-way, with both an upload and a download. However, you can also choose to perform an upload-only synchronization or a download-only synchronization.

- **File-based download** Downloads can be distributed as files, enabling offline distribution of synchronization changes. This feature includes functionality to ensure that the correct data is applied.

- **Server-initiated synchronization** You can initiate MobiLink synchronization from the consolidated database. This means you can push data updates to remote databases, and cause remote databases to upload data to the consolidated database. See "MobiLink - Server-Initiated Synchronization".

- **Choice of network protocols** Synchronization can occur over TCP/IP, HTTP, or HTTPS. Windows Mobile devices can synchronize using Microsoft ActiveSync.

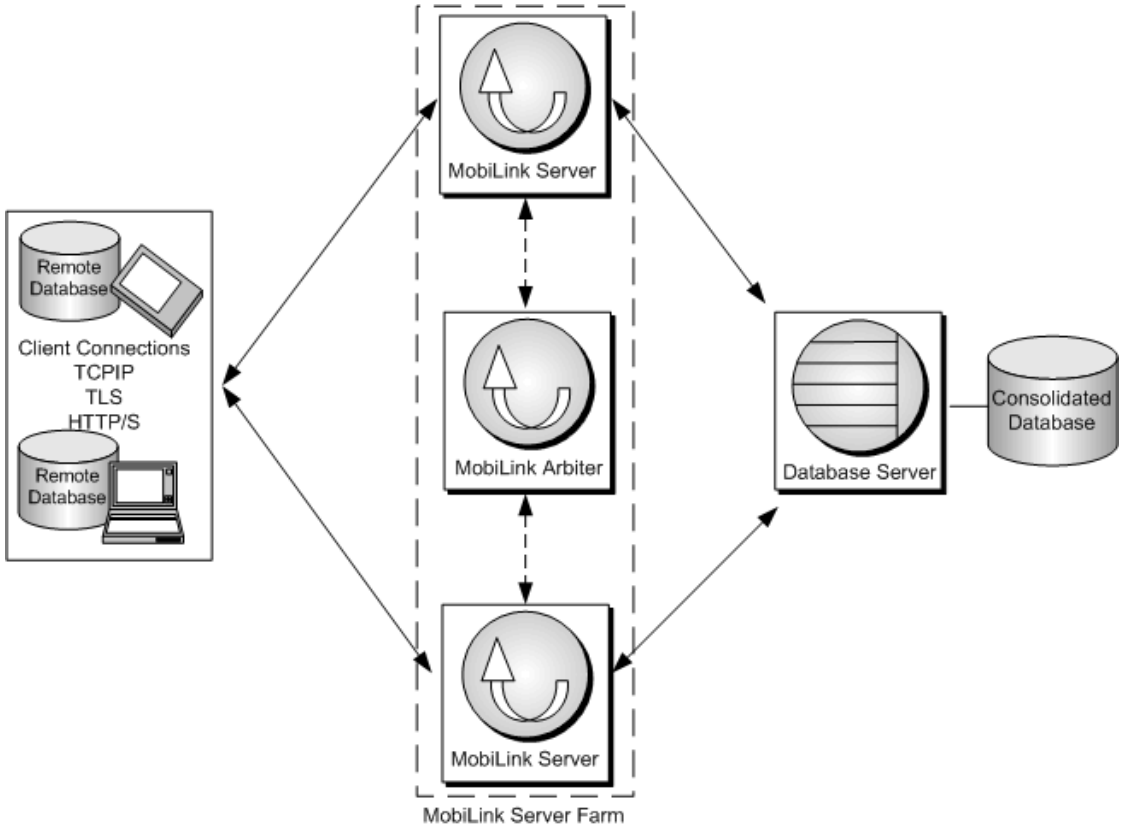
- **Session-based** All changes can be uploaded in a single transaction and downloaded in a single transaction. At the end of each successful synchronization, the consolidated and remote databases are consistent. (If you want to preserve the order of transactions, you can also choose to have each transaction on the remote database uploaded as a separate transaction.)

  Either a whole transaction is synchronized, or none of it is synchronized. This ensures transactional integrity for each database.

- **Data consistency** MobiLink operates using a loose consistency policy. All changes are synchronized with each site over time in a consistent manner, but different sites may have different copies of data at any instant.

- **Wide variety of hardware and software platforms** A variety of widely-used database management systems can be used as a MobiLink consolidated database, or you can define synchronization to an arbitrary data source using the MobiLink server API. Remote databases can be SQL Anywhere or UltraLite. The MobiLink server runs on Windows, Unix, Linux, and Mac OS X. SQL Anywhere runs on Windows, Windows Mobile, or Unix, Linux, and Mac OS X. UltraLite runs on Windows Mobile or BlackBerry. See "Supported platforms" [*SQL Anywhere 12 - Introduction*].

- **MobiLink arbiter** A MobiLink arbiter ensures that only a single MobiLink server in a server farm is running as the primary server. This prevents redundant notifications in a server-initiated synchronization environment and preserves messages in a QAnywhere messaging environment. The diagram below shows the MobiLink arbiter in a server farm environment.



Quick start to MobiLink

MobiLink is designed to synchronize data among many remote applications that connect intermittently with one or more central data sources. In a basic MobiLink application, your remote clients are SQL Anywhere or UltraLite databases, and your central data source is one of the supported ODBC-compliant relational databases. This architecture can be extended using the MobiLink server API so that there are virtually no restrictions on what you synchronize to on the server side.

In all MobiLink applications, the MobiLink server is the key to the synchronization process. Synchronization typically begins when a MobiLink remote site opens a connection to a MobiLink server. During synchronization, the MobiLink client at the remote site can upload database changes that were made to the remote database since the previous synchronization. On receiving this data, the MobiLink server updates the consolidated database, and then can download changes from the consolidated database to the remote database.

The quickest way to start developing a MobiLink application is to use the **Create Synchronization Model Wizard**. When you use the wizard, most of the steps outlined below are handled for you. See "Introduction to synchronization models" on page 24.

However, even when using a MobiLink model, you need to understand the process and components of MobiLink synchronization.

Overview of a MobiLink application

To create a MobiLink application

1. Set up a consolidated database.
   - Run setup scripts against the database to add system objects required by MobiLink synchronization. Alternatively, you can create a separate system database to hold these objects.

     See "MobiLink consolidated databases" [*MobiLink - Server Administration*].
2. Set up remote databases.
   - Your remote databases can be SQL Anywhere, UltraLite, or a combination of the two.
   - In your remote databases, create MobiLink users. See "MobiLink users" [*MobiLink - Client Administration*].
   - To determine the upload in a SQL Anywhere remote database, create publications and subscriptions. See "Publishing data" [*MobiLink - Client Administration*].

     To determine the upload in an UltraLite remote database, create publications. See "Publications in UltraLite" [*UltraLite - Database Management and Reference*].
3. To determine how the upload is applied, create server synchronization logic. See "Writing synchronization scripts" [*MobiLink - Server Administration*].
4. To download data that has changed since the last download, set up timestamp-based synchronization. See "Timestamp-based downloads" [*MobiLink - Server Administration*].
5. Start the MobiLink server. See "Running the MobiLink server" [*MobiLink - Server Administration*].
6. Initiate synchronization on the client.
   - For SQL Anywhere remote databases, see "Initiating synchronization" [*MobiLink - Client Administration*].
   - For UltraLite remote databases, see "Designing synchronization in UltraLite" [*UltraLite - Database Management and Reference*].

Introductory reading

- "Understanding MobiLink synchronization" on page 1
- "Synchronization techniques" [*MobiLink - Server Administration*]

Tutorials

- "Introduction to MobiLink synchronization" on page 73
- "Exploring the CustDB sample for MobiLink" on page 46
- "UltraLite CustDB samples" [*UltraLite - Database Management and Reference*]
- "Exploring the MobiLink Contact sample" on page 60
- "Tutorial: Using MobiLink with an Oracle Database 10g" on page 92
- "Tutorial: Using MobiLink with an Adaptive Server Enterprise consolidated database" on page 107
- "Tutorial: Using Java synchronization logic" on page 123
- "Tutorial: Using .NET synchronization logic" on page 131
- "Tutorial: Using Java or .NET for custom user authentication" on page 141
- "Tutorial: Introduction to direct row handling" on page 148
- "Tutorial: Synchronizing with Microsoft Excel" on page 171
- "Tutorial: Synchronizing with XML" on page 189

Other resources for getting started

- MobiLink provides samples that you can examine and run to explore MobiLink functionality. MobiLink samples are installed with the product in *samples-dir\MobiLink*. (For the location of *samples-dir* on your operating system, see "Samples directory" [*SQL Anywhere Server - Database Administration*].)

- MobiLink code exchange samples are located at http://www.sybase.com/detail?id=1058600#319.

Designing a MobiLink application

There are two basic architectures for database applications:

- **Online applications** Users update data by connecting to the central database directly. When a connection is unavailable, the user cannot work.

- **Occasionally connected smart client applications** Each user has a local database. Their database application is always available to them, regardless of connectivity, and is kept synchronized with other databases in the system.

MobiLink is designed for creating occasionally connected smart client applications. Smart client applications can greatly increase the usability, efficiency, and scalability of an application, but they pose new issues for application developers. This section describes some of the major issues facing developers of smart client applications, and describes how you can implement solutions in a MobiLink synchronization environment.

Synchronize only what you need

In most applications it would be a disaster to download the entire consolidated database every time you want to update any piece of data on your remote device. The time and bandwidth would be prohibitive, making the whole system unworkable. There are various techniques for ensuring you upload and download only what each user needs.

First, each remote database should only contain a subset of the tables and columns in the consolidated database. For example, a salesperson in Region A may need different tables and columns than a salesperson in Region B or a supervisor.

Of the tables and columns that you put on a remote device, you only want to mark ones for synchronization that need to be synchronized. In a MobiLink application you can map tables and columns, regardless of their names, as long as the data types match. By default, data is both uploaded and downloaded, but MobiLink also allows you to specify that certain columns are upload-only or download-only.

Your synchronization should only download rows to a remote database that are relevant to the user. You might want to partition your download by remote database, by user, or by other criteria. For example, a sales rep in Region A may only need data updates about Region A.

You only want to update data that has changed. In a MobiLink application the upload is based on the transaction log and so by default, data is only uploaded if it has changed on the remote database. To do the same for the download, you specify timestamp-based synchronization so that your system records the time that data is successfully downloaded, and data is downloaded only if it has changed since then.

You may also want to implement a system of high priority synchronization: time-sensitive data is scheduled to be updated frequently, but less time-critical data is scheduled to be updated at night or when the device is in a cradle. You can implement high-priority synchronization by creating different publications that are scheduled to run at different times.

In addition, your users may benefit from a push-synchronization system, where data is effectively pushed down to remote devices when needed. For example, if a trucking company dispatcher learns of a traffic disruption, they can download an update to the truck drivers who are heading towards that area. In MobiLink, this is called server-initiated synchronization.

Handle upload conflicts

Say you have a warehouse. Each employee has a handheld device that they use to update inventory as they add or remove boxes. They start a shift with 100 boxes, so each employee's remote database registers 100, as does the consolidated database. David removes 20 boxes. He updates his database and synchronizes. Now both his database and the consolidated database register 80. Now Susan removes ten boxes. But when Susan updates her database and synchronizes, her application expects the consolidated database to have 100 boxes, not 80. This generates an upload conflict.

In this warehouse application, the solution is to create conflict resolution logic that says that the correct value is whatever David updated it to, minus the original value less Susan's value:

```
80 - (100 - 90) = 70
```

While this conflict resolution logic works for inventory-based applications such as a warehouse, it isn't appropriate in all business applications. With MobiLink, you can define conflict resolution logic to cover:

- **Inventory model** Update the row for the correct number of units.
- **Date** The latest update wins (based on when the value was changed in the database, not when the value was synchronized).

- **Person** For example, the manager always wins or the owner of the record always wins.
- **Custom** Just about any other business logic you need to implement.

Sometimes you can design your system so that upload conflicts cannot occur. If data is partitioned on the remotes so that there is no overlap, conflicts may be avoided. However, if conflicts can happen, you should create a programmatic solution for detecting and resolving them.

Unique primary keys

To upload data, detect upload conflicts, and synchronize deleted rows on the consolidated database, you must have unique primary keys on every synchronized table in your database system. Each row must have a primary key that is unique not only within the database, but within the entire database system. Primary keys must not be updated.

MobiLink provides several ways to guarantee unique primary keys. One is to set the data type of the primary key to a GUID. GUID, which stands for Globally Unique Identifier, is a 16-byte hexadecimal number. MobiLink provides a NEWID function that causes a GUID to be created automatically for a new row.

Another solution is a composite key. In MobiLink, each remote database has a unique value called a remote ID. Your primary keys could be formed from the remote ID plus a regular primary key, such as an ordinal value.

SQL Anywhere also offers a global autoincrement solution. You declare a column as global autoincrement and then when a row is added, the primary key is automatically created by incrementing the last value. This solution works best when your consolidated database is SQL Anywhere.

Finally, you can create a pool of primary key values that are distributed to remote databases.

How you choose which primary key system to use, like many decisions in developing a synchronization solution, has to do with the level of control you have over the consolidated and remote databases. Often, the remote databases must be able to operate without any administration. You may also find that it is difficult to change the schema on the consolidated database. In addition, your choice of RDBMS for the consolidated database may limit your options, as not all RDBMSs support all features.

Handling deletes

Another issue in a synchronization system is how to handle rows that are deleted from the consolidated database. Say I delete a row from the consolidated database. The next time David synchronizes his remote database, the delete is downloaded—deleting the row from David's database. But what do I do with it on the consolidated database? I can't delete it because I need to download the delete to Susan as well.

Here are two ways you can handle download deletes. First, you can add a status column to each table that indicates whether the row is deleted or not. In this case, the row is never deleted—it is just marked for deletion. You can occasionally clean up the rows marked for deletion, once you are sure that all the remote databases are up to date. Alternatively, you can create a shadow table for each table. The shadow table stores the primary key values of deleted rows. When a row is deleted, a trigger populates the shadow table, and the values in the shadow table determine what to delete on the remote database.

Transactions

In a synchronized database system, only database transactions that are committed should be synchronized. In addition, all committed transactions involving data that is to be synchronized should be synchronized, or an error should be generated. This is the default behavior in MobiLink.

You also need to consider the isolation level of the connection to the consolidated database. You need to use an isolation level that provides the best performance possible while ensuring data consistency. Isolation level 0 (READ UNCOMMITTED) is generally unsuitable for synchronization as it can lead to inconsistent data.

By default, MobiLink uses the isolation level SQL_TXN_READ_COMMITTED for uploads, and if possible it uses snapshot isolation for downloads (otherwise it uses SQL_TXN_READ_COMMITTED). Snapshot isolation eliminates the problem of downloads being blocked until transactions are closed on the consolidated database, but not all RDBMSs support it.

Daylight savings time

The annual change to daylight savings time can pose a problem for synchronized databases during the hour that the time changes. In the autumn the time moves back an hour; 2:00 AM becomes 1:00 AM. If you attempt to synchronize between 1:00 AM and 2:00 AM, the timestamp of the synchronization is ambiguous: is it the first 1:15 AM or the second 1:15 AM?

To resolve this problem you can shut down for an hour when the time changes in the autumn, or you can put your consolidated database server on coordinated universal time (UTC) time.

Further reading

- "Synchronization techniques" [*MobiLink - Server Administration*]

Options for developing a MobiLink application

MobiLink provides a variety of ways to develop an application. You can use these methods in alone or in combination.

- **Create Synchronization Model Wizard** The wizard walks you through the development of your application. You start with a central database that has schema, and you can create remote databases and the scripts needed for synchronization. The wizard can also create shadow tables on your consolidated database to handle things like download deletes. When the wizard completes, you can further customize the model. There is a **Deploy Synchronization Model Wizard** that creates databases and tables, updates the MobiLink system tables, and creates scripts that run MobiLink utilities.

  Once you have deployed a MobiLink model, if you have further customizations to it you can still make changes using one of the methods described below.

  See "MobiLink Plug-in for Sybase Central" on page 20.

- **Sybase Central** The MobiLink 12 plug-in for Sybase Central enables you to update all the elements of your MobiLink application.

See "Synchronization model tasks" on page 26.

- **System procedures** When you set up a central database to operate as a consolidated database, system objects are created that are used by MobiLink synchronization. These include MobiLink system tables, where the server side of your MobiLink application is largely stored. They also include system procedures and utilities that you can use to insert MobiLink scripts into your MobiLink system tables, register remote users, and so on. See:
  - "MobiLink server system procedures" [*MobiLink - Server Administration*]
  - "MobiLink utilities" [*MobiLink - Server Administration*]
- **Direct manipulation of MobiLink system tables** Advanced users may want to add, delete, and update data in the MobiLink system tables directly. Doing so requires an advanced understanding of how MobiLink works.

  See "MobiLink server system tables" [*MobiLink - Server Administration*].

Options for writing server-side synchronization logic

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

SQL synchronization logic is usually best when synchronizing to a supported consolidated database.

Java and .NET are useful if you are synchronizing against something other than a supported consolidated database. They may also be useful if your design is restricted by the limitations of the SQL language or by the capabilities of your database management system, or if you simply want portability across different RDBMS types.

Java and .NET synchronization logic can function just as SQL logic functions. The MobiLink server can make calls to Java or .NET methods on the occurrence of MobiLink events just as it can access SQL scripts on the occurrence of MobiLink events. When you are working in Java or .NET, you can use the events to do some extra processing, but when you are processing scripts for events that directly handle upload or download rows, your implementation must return a SQL string. With the exception of the two events used in direct row handling, uploads and downloads are not directly accessible from Java or .NET synchronization logic: MobiLink executes the string returned by Java or .NET as SQL.

Direct row handling, which uses the events handle_UploadData and handle_DownloadData to synchronize against a data source, **does** directly manipulate the upload and download rows.

The following are some scenarios where you might want to consider writing scripts in Java or .NET:

- **Direct row handling** With Java and .NET synchronization logic, you can use MobiLink to access data from data sources other than your consolidated database, such as application servers, web servers, and files.
- **Authentication** A user authentication procedure can be written in Java or .NET so that MobiLink authentication integrates with your corporate security policies.

- **Stored procedures** If your RDBMS lacks the ability to use user-defined stored procedures, you can create a method in Java or .NET.

- **External calls** If your program calls for contacting an external server midway through a synchronization event, you can use Java or .NET synchronization logic to perform actions triggered by synchronization events. Java and .NET synchronization logic can be shared across multiple connections.

- **Variables** If your database lacks the ability to handle variables, you can create a variable in Java or .NET that persists throughout your connection or synchronization. (Alternatively, with SQL scripts you can use user-defined named parameters, which work with all consolidated database types. See "User-defined named parameters" [*MobiLink - Server Administration*].)

MobiLink server APIs

Java and .NET synchronization logic are available via the MobiLink server APIs. The MobiLink server APIs are sets of classes and interfaces for MobiLink synchronization.

The MobiLink server API for Java offers you:

- Access to the existing ODBC connection to the consolidated database as a JDBC connection.

- Access to alternate data sources using interfaces such as JDBC, web services, and JNI.

- The ability to create new JDBC connections to the consolidated database to make database changes outside the current synchronization connection. For example, you can use this for error logging or auditing, even if the synchronization connection does a rollback.

- For synchronizing with the consolidated database, the ability to write and debug Java code before it is executed by the MobiLink server. SQL development environments for many database management systems are relatively primitive compared to those available for Java applications.

- Both SQL row handling and direct row handling.

- The full richness of the Java language and its large body of existing code and libraries.

See "MobiLink server API for Java reference" [*MobiLink - Server Administration*].

The MobiLink server API for .NET offers you:

- Access to the existing ODBC connection to the consolidated database using iAnywhere classes that call ODBC from .NET.

- Access to alternate data sources using interfaces such as ADO.NET, web services, and OLE DB.

- For synchronizing with the consolidated database, the ability to write and debug .NET code before it is executed by the MobiLink server. SQL development environments for many database management systems are relatively primitive compared to those available for .NET applications.

- Both SQL row handling and direct row handling.

- Code that runs inside the .NET Common Language Runtime (CLR) and allows access to all .NET libraries, including both SQL row handling and direct row handling.

See "MobiLink server .NET API reference (.NET 2.0)" [*MobiLink - Server Administration*].

See also

- "Writing synchronization scripts" [*MobiLink - Server Administration*]
- "Synchronization techniques" [*MobiLink - Server Administration*]
- "Writing synchronization scripts in Java" [*MobiLink - Server Administration*]
- "Writing synchronization scripts in .NET" [*MobiLink - Server Administration*]
- "Direct row handling" [*MobiLink - Server Administration*]

The synchronization process

A **synchronization** is a process of data exchange between MobiLink clients and a central data source. During this process, the client must establish and maintain a session with the MobiLink server. If successful, the session leaves the remote and consolidated databases in a mutually consistent state.

The client normally initiates the synchronization process. It begins by establishing a connection to the MobiLink server.

The upload and the download

To upload rows, MobiLink clients prepare and send an **upload** that contains a list of all the rows that have been updated, inserted, or deleted on the remote database since the last synchronization. Similarly, to download rows, the MobiLink server prepares and sends a **download** that contains a list of inserts, updates, and deletes.

- **Upload** By default, the MobiLink client automatically keeps track of which rows in the remote database have been inserted, updated, or deleted since the last successful synchronization. Once the connection is established, the MobiLink client uploads a list of all these changes to the MobiLink server.

  The upload consists of a set of new and old row values for rows modified in the remote database. (Updates have new and old row values; deletes have old values; and inserts have new values.) If a row has been updated or deleted, the old values are those that were present immediately following the last successful synchronization. If a row has been inserted or updated, the new values are the current row values. No intermediate values are sent, even if the row was modified several times before arriving at its current state.

  The MobiLink server receives the upload and executes upload scripts that you define. By default it applies all the changes in a single transaction. When it has finished, the MobiLink server commits the transaction.

- **Download** The MobiLink server compiles a list of rows to insert, update, or delete on the MobiLink client, using synchronization logic that you create. It downloads these rows to the MobiLink client. To compile this list, the MobiLink server opens a new transaction on the consolidated database.

The MobiLink client receives the download. It takes the arrival of the download as confirmation that the consolidated database has successfully applied all uploaded changes. It ensures that these changes are not sent to the consolidated database again.

Next, the MobiLink client automatically processes the download, deleting old rows, inserting new rows, and updating rows that have changed. It applies all these changes in a single transaction in the remote database. When finished, it commits the transaction.

During MobiLink synchronization, there are few distinct exchanges of information. The client builds and uploads the entire upload. In response, the MobiLink server builds and downloads the entire download. It is important to limit the verbosity of the protocol when communication is slower and has higher latency, such as when using telephone lines or public wireless networks.

Note
MobiLink operates using the ODBC isolation level SQL_TXN_READ_COMMITTED as the default isolation level for the consolidated database. If the RDBMS used for the consolidated database supports snapshot isolation, and if snapshot is enabled for the database, then by default MobiLink uses snapshot isolation for downloads. See "MobiLink isolation levels" [*MobiLink - Server Administration*].

See also

- "Overview of MobiLink events" [*MobiLink - Server Administration*]
- "Events during upload" [*MobiLink - Server Administration*]
- "Events during download" [*MobiLink - Server Administration*]

MobiLink events

When the MobiLink client initiates a synchronization, several synchronization events occur. At the occurrence of a synchronization event, MobiLink looks for a script to match the event. The script contains instructions detailing what you want done. If you have defined a script for the event and put it in a MobiLink system table, it is invoked.

MobiLink scripts

Whenever an event occurs, the MobiLink server executes the associated script if you have created one. If no script exists, the next event in the sequence occurs.

Note
When you use the **Create Synchronization Model Wizard** to create your MobiLink application, all the required MobiLink scripts are created for you. However, you can customize the default scripts, including creating new scripts.

The following are the typical upload scripts for tables. The first event, upload_insert, triggers the running of the upload_insert script, which inserts any changes in the emp_id and emp_name columns into the emp table. The upload_delete and upload_update scripts perform similar functions for delete and update actions on the emp table.

Event	Example script contents
upload_insert	`INSERT INTO emp (emp_id,emp_name) VALUES ({ml r.emp_id}, {ml r.emp_name})`
upload_delete	`DELETE FROM emp WHERE emp_id = {ml r.emp_id}`
upload_update	`UPDATE emp SET emp_name = {ml r.emp_name} WHERE emp_id = {ml r.emp_id}`

The download script uses a cursor. The following is an example of a download_cursor script:

```
SELECT order_id, cust_id
FROM ULOrder
WHERE last_modified >= {ml s.last_table_download}
AND emp_name = {ml r.emp_id}
```

For more information about events and scripts, see:

- "Writing synchronization scripts" [*MobiLink - Server Administration*]
- "Synchronization events" [*MobiLink - Server Administration*]

Scripts can be written in SQL, Java, or .NET

You can write scripts using the native SQL dialect of your consolidated database, or using Java or .NET synchronization logic. Java and .NET synchronization logic allow you to write code, invoked by the MobiLink server, to connect to a database, manipulate variables, directly manipulate uploaded row operations, or add row operations to the download. There is a MobiLink server API for Java and a MobiLink server API for .NET that provide classes and methods to suit the needs of synchronization.

See "Options for writing server-side synchronization logic" on page 11.

For information about RDBMS-dependent scripting, see "MobiLink consolidated databases" [*MobiLink - Server Administration*].

Storing scripts

SQL scripts are stored in MobiLink system tables in the consolidated database. For scripts written with the MobiLink server APIs, you store the fully qualified method name as the script. You can add scripts to a consolidated database in several ways:

- When you use the **Create Synchronization Model Wizard**, scripts are stored in the MobiLink system tables when you deploy your project.
- You can manually add scripts to the system tables by using stored procedures that are installed when you set up a consolidated database.
- You can manually add scripts to the system tables using Sybase Central.

See "Adding and deleting scripts" [*MobiLink - Server Administration*].

Transactions in the synchronization process

The MobiLink server incorporates changes uploaded from each MobiLink client into the consolidated database in one transaction. It commits these changes once it has completed inserting new rows, deleting old rows, making updates, and resolving any conflicts.

Caution
There should be no implicit or explicit commit or rollback in your SQL synchronization scripts or the procedures or triggers that are called from your SQL synchronization scripts. COMMIT or ROLLBACK statements within SQL scripts alter the transactional nature of the synchronization steps. If you use them, MobiLink cannot guarantee the integrity of your data in the event of a failure.

Tracking downloaded information

MobiLink uses a last download timestamp, stored in the remote database, to help simplify how downloads are created.

The primary role of the download transaction is to select rows in the consolidated database. If the download fails, the remote uploads the same last download timestamp over again, and no data is lost.

See "Using last download times in scripts" [*MobiLink - Server Administration*].

Begin and end transactions

The MobiLink client processes information in the download in one transaction. Rows are inserted, updated, and deleted to bring the remote database up to date with the consolidated data.

The MobiLink server uses two other transactions, one at the beginning of synchronization and one at the end. These transactions allow you to record information regarding each synchronization and its duration. So, you can record statistics about attempted synchronizations, successful synchronizations, and the duration of synchronizations. Since data is committed at various points in the process, these transactions also let you commit data that can be useful when analyzing failed synchronization attempts.

See also

- "Overview of MobiLink events" [*MobiLink - Server Administration*]
- "Events during upload" [*MobiLink - Server Administration*]
- "Events during download" [*MobiLink - Server Administration*]

How synchronization failure is handled

MobiLink synchronization is fault tolerant. For example, if a communication link fails during synchronization, both the remote database and the consolidated database are left in a consistent state.

On the client, failure is indicated by a return code.

Synchronization failure is handled differently depending on when it happens. The following cases are handled in different ways:

- **Failure during upload** If the failure occurs while building or applying the upload, the remote database is left in exactly the same state as at the start of synchronization. At the server, any part of the upload that has been applied is rolled back.

- **Failure between upload and download** If the failure occurs once the upload is complete, but before the MobiLink client receives the download, the client cannot be certain whether the uploaded changes were successfully applied to the consolidated database. The upload might be fully applied and committed, or the failure may have occurred before the server applied the entire upload. The MobiLink server automatically rolls back incomplete transactions in the consolidated database.

  The MobiLink client maintains a record of all uploaded changes. The next time the client synchronizes, it requests the state of the previous upload before building the new upload. If the previous upload was not committed, the new upload contains all changes from the previous upload.

- **Failure during download** When a failure occurs on the remote device while applying the download, any part of the download that has been applied is rolled back and the remote database is left in the same state as before the download.

  If you are using non-blocking download acknowledgement, the download transaction has already been committed, but neither the nonblocking_download_ack script nor the publication_nonblocking_download_ack script is invoked.

  If you are not using download acknowledgement, there is no server-side consequence of a download failure.

No data is lost when a failure occurs. The MobiLink server and the MobiLink client manage this for you. Neither you nor the user need to worry about maintaining consistent data in their application.

How the upload is processed

When the MobiLink server receives an upload from a MobiLink client, the entire upload is stored until the synchronization is complete. This is done for the following reasons:

- **Filtering download rows** The most common technique for determining which rows to download is to download rows that have been modified since the most recent download. When synchronizing, the upload precedes the download. Any rows that are inserted or updated during the upload are rows that have been modified since the previous download.

  It would be difficult to write a download_cursor script that omits from the download rows that were sent as part of the upload. For this reason, the MobiLink server automatically removes these rows from the download.

- **Processing inserts and updates** By default, tables in the upload are applied to the consolidated database in an order that avoids referential integrity violations. The tables in the upload are ordered based on foreign key relationships. For example, if table A and table C both have foreign keys that reference a primary key column in B, then inserts and updates for table B rows are uploaded first.

- **Processing deletes after inserts and updates** Deletes are applied to the consolidated database after all inserts and updates are applied. When deletes are applied, tables are processed in the opposite

order from the way they appear in the upload. When a row being deleted references a row in another table that is also being deleted, this order of operations ensures that the referencing row is deleted before the referenced row is deleted.

- **Deadlock** When an upload is being applied to the consolidated database, it may encounter deadlock due to concurrency with other transactions. These transactions might be upload transactions from other MobiLink server database connections, or transactions from other applications using the consolidated database. When an upload transaction is deadlocked, it is rolled back and the MobiLink server automatically starts applying the upload again, from the beginning.

Performance tip
It is important to write your synchronization scripts to avoid contention as much as possible. Contention has a significant impact on performance when multiple users are synchronizing simultaneously.

Referential integrity and synchronization

All MobiLink clients, with the exception of UltraLiteJ, enforce referential integrity when they incorporate the download into the remote database.

Rather than failing the download transaction, by default the MobiLink client automatically deletes all rows that violate referential integrity.

This feature has the following benefits:

- Protection from mistakes in your synchronization scripts. Given the flexibility of the scripts, it is possible to accidentally download rows that would break the integrity of the remote database. The MobiLink client automatically maintains referential integrity without requiring intervention.

- You can use this referential integrity mechanism to delete information from a remote database efficiently. By only sending a delete to a parent record, the MobiLink client removes all the child records automatically for you. This can greatly reduce the amount of traffic MobiLink must send to the remote database.

MobiLink clients provide notification if they have to explicitly delete rows to maintain referential integrity, as follows:

- For SQL Anywhere clients, dbmlsync writes an entry in the log. There are also dbmlsync event hooks that you can use. See:

  - "sp_hook_dbmlsync_download_ri_violation" [*MobiLink - Client Administration*]
  - "sp_hook_dbmlsync_download_log_ri_violation" [*MobiLink - Client Administration*]

- For UltraLite clients, the warning SQLE_ROW_DELETED_TO_MAINTAIN_REFERENTIAL_INTEGRITY is raised. This warning takes a parameter which is the table name. To maintain referential integrity, the warning is raised on every row that is deleted. Your application can ignore the warnings if you want synchronization to

proceed. If you want to explicitly handle the warnings, you can use the error callback function to trap them and, for example, count the number of rows that are deleted.

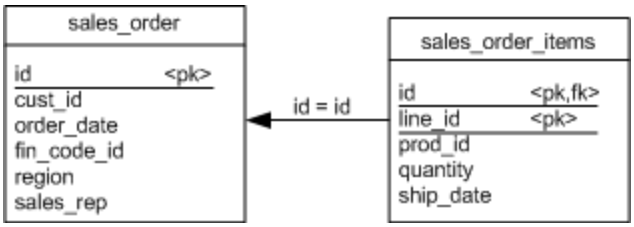
If you want synchronization to fail when the warning is raised, you must implement a synchronization observer and then signal the observer (perhaps through a global variable) from the error callback function. In this case, synchronization fails on the next call to the observer.

Referential integrity checked at the end of the transaction

The MobiLink client incorporates changes from the download in a single transaction. To offer more flexibility, referential integrity checking occurs at the end of this transaction. Because checking is delayed, the database may temporarily pass through states where referential integrity is violated. This is because rows that violate referential integrity are automatically removed before the download is committed.

Example

Suppose that an UltraLite sales application contains the following two tables. One table contains sales orders. Another table contains items that were sold in each order. They have the following relationship:



If you use the download_delete_cursor for the sales_order table to delete an order, the default referential integrity mechanism automatically deletes all rows in the sales_order_items table that point to the deleted sales order.

This arrangement has the following advantages:

- You do not require a sales_order_items table script because rows from this table are deleted automatically.
- The efficiency of synchronization is improved. You need not download rows to delete from the sales_order_items table. If each sales order contains many items, the performance improves because the download is now smaller. This technique is particularly valuable when using slow communication methods.

Changing the default behavior

For SQL Anywhere clients, you can use the sp_hook_dbmlsync_download_ri_violation client event hook to handle the referential integrity violation. Dbmlsync also writes an entry to its log.

See:

- "sp_hook_dbmlsync_download_log_ri_violation" [*MobiLink - Client Administration*]
- "sp_hook_dbmlsync_download_ri_violation" [*MobiLink - Client Administration*]

Security

There are several aspects to securing data throughout a widely distributed system such as a MobiLink installation:

- **Protecting data in the consolidated database** Data in the consolidated database can be protected using the database user authentication system and other security features.

  For more information, see your database documentation. If you are using a SQL Anywhere consolidated database, see "Keeping your data secure" [*SQL Anywhere Server - Database Administration*].

- **Protecting data in the remote databases** If you are using SQL Anywhere remote databases, the data can be protected using SQL Anywhere security features. By default, these features are designed to prevent unauthorized access through client/server communications, but should not be considered a sure-fire method pf preventing a serious attempt to extract information directly from the database file.

  Files on the client are protected by the security features of the client operating system.

  If you are using a SQL Anywhere remote database, see "Keeping your data secure" [*SQL Anywhere Server - Database Administration*].

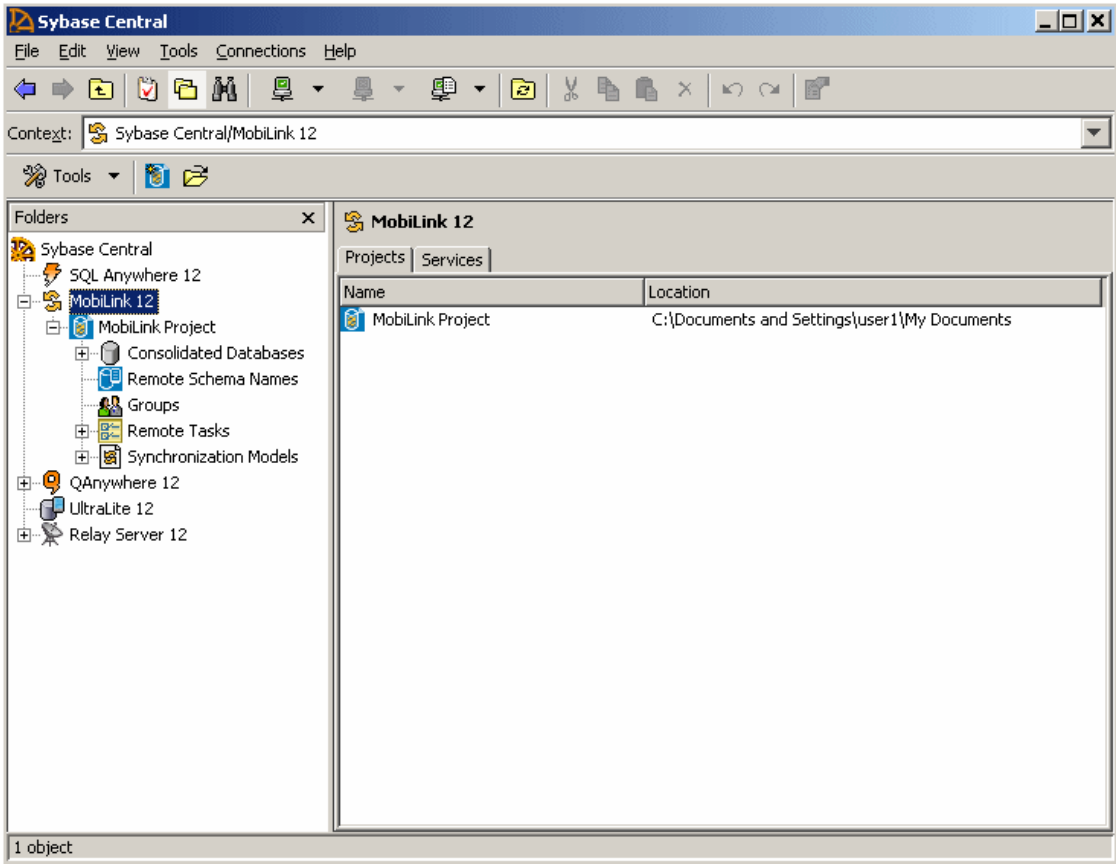
  If you are using an UltraLite database, see "Securing UltraLite databases" [*UltraLite - Database Management and Reference*].

- **Protecting data during synchronization** Communication from MobiLink clients to MobiLink servers can be protected by the MobiLink transport layer security features. See "Transport-layer security" [*SQL Anywhere Server - Database Administration*].

- **Protecting the synchronization system from unauthorized users** MobiLink synchronization can be secured by a password-based user authentication system. This mechanism prevents unauthorized users from synchronizing data. See "MobiLink users" [*MobiLink - Client Administration*].

MobiLink Plug-in for Sybase Central

The MobiLink plug-in for Sybase Central has been redesigned in version 12. In previous versions, the plug-in had two modes: model mode and admin mode. The MobiLink functionality was split between these two modes, so you needed to be aware of which mode you were in at any given time. In version 12, these modes no longer exist, as show below.



The top level functions available through the **Folders** pane of the MobiLink plug-in are:

- Working with MobiLink projects. See "Creating a MobiLink project" on page 22.
- Working with consolidated databases. See "Adding a consolidated database" on page 23.
- Working with remote schema names. See "Working with Agents in Sybase Central" [*MobiLink - Server Administration*].
- Working with groups. See "Working with Agents in Sybase Central" [*MobiLink - Server Administration*].
- Working with remote tasks. "Working with remote tasks" [*MobiLink - Server Administration*].
- Working with synchronization models. See "Introduction to synchronization models" on page 24.

Remote schema names, groups and remote tasks are all part of the central administration of remote databases feature. See "Central administration of remote databases" [*MobiLink - Server Administration*].

To start working with MobiLink in Sybase Central, you must first define a **MobiLink Project**.

A **MobiLink project** is a framework that allows you to work with MobiLink objects in Sybase Central. You cannot do anything in the MobiLink 12 plug-in without having a MobiLink project. The MobiLink project enables you to schedule and monitor tasks, even if you are not creating a synchronization model, and you must have a project to create a synchronization model.

A MobiLink project is a named collection of the following:

- A list of synchronization models
- A list of designed but undeployed remote tasks
- A list of connections to a consolidated database
- A list of MobiLink users
- A list of user-defined groups, remote databases or devices
- A list of remote databases
- A list of notifiers

Creating a MobiLink project

Use the following procedure to create a MobiLink project in Sybase Central.

To create a MobiLink project

1. In the left pane, right-click **MobiLink 12** » **New** » **Project**.
2. In the **Name** field, type a name for the project.
3. In the **Location** field, type the location of the project folder or click **Browse** to select the folder for your project file.
4. Click **Finish** to save your project with just the name and location, or click **Next** to specify additional information in the following steps.
5. If you clicked **Next**, click **Add A Consolidated Database To The Project** if you know which consolidated database you want to associate with the project.
   a. In the **Database Display Name** field, type the display name you want to use for the consolidated database. This is the name that is listed in the consolidated database list in your project.
   b. In the **Connection String** field, type the database connection parameters to use to connect to the consolidated database or click **Edit** to go open a dialog to connect to an ODBC data source.
   c. Select **Remember The Password** if you want to save the password you used to connect to the database.
   d. Click **Next**.

6. Decide whether you want to add a synchronization model or create one later. Choose one of the following options:
   - **Do Not Add A Model** Choose this option if you do not want to add or create a synchronization model now or you can add one later.
   - **Create A New Model** Choose this option to create a new synchronization model. The **Create Synchronization Model Wizard** is launched when you finish creating the project. See "Introduction to synchronization models" on page 24.
   - **Import The Model From The Following File** Choose this option to import an existing synchronization model. In the empty field, type the path and file name of the synchronization model to import, or click **Browse** to select the synchronization model file. This creates a copy of the synchronization model file in your project folder.
7. Select **Add A Remote Schema Name To The Project** if you want to identify a group of remote databases that have the same schema or you can add one later. If you are adding a remote schema name, specify the following:
   - **What Do You Want To Name The New Remote Schema Name?** Type the name you want to use to identify the group of remote databases that share the same schema. It might be a good idea to include a version number.
   - **Which Type Of Database Does The Remote Schema Name Apply To?** Select the type of database that the remote schema name applies to. This can be either **SQL Anywhere** or **UltraLite**.
8. Click **Finish** to save the new project.

Adding a consolidated database

You can add one or more consolidated databases to a MobiLink project in Sybase Central. A remote task must have at least one consolidated database assigned to it before it can be deployed.

To add a consolidated database

1. Ensure the MobiLink project you are working with is selected in the **Folders** view in the left pane, then right-click on the project name and choose **New** » **Consolidated Database**.
2. Enter the required database connection parameters and click **Next** to connect to the database.
3. In the **Display Name** field, type the name you want to use for this database in your project. The default display name is the ODBC data source name and if you want to provide a description of the database, type it in the **Description** field.
4. Select **Remember The Password** if you want to save the password you used to connect to the database.
5. Click **Finish** to add the consolidated database to the project.

Checking MobiLink system setup

You must add objects such as tables, columns, and triggers that are required for synchronization before you can use a database as a MobiLink consolidated database. You add these objects by running a setup script against the database. There is a separate setup script for each supported RDBMS. These scripts are all located in the *install-dir\MobiLink\setup* folder. You can verify exactly what the script does by opening it in a text editor.

When you add a consolidated database to your MobiLink project, it checks the MobiLink system setup. If it is missing, you are prompted to install the MobiLink system setup, or you can install it later. You may also be prompted to upgrade if the check found an older version. You are also prompted to install the MobiLink system setup when you deploy a synchronization model, if you have not already done so.

To check MobiLink system setup

1. Ensure the MobiLink project you are working with is selected in the **Folders** view in the left pane, then right-click the consolidated database you want to check and choose **Check MobiLink System Setup**.

2. If setup is not already installed or is not up to date, click **Yes** to install it or update it, then click **OK**. You may also be prompted to upgrade if the check found an earlier version.

See also

- "Setting up a consolidated database" [*MobiLink - Server Administration*]
- "Deploying synchronization models" on page 38

Introduction to synchronization models

A **synchronization model** is a tool that makes it easy for you to create MobiLink applications. A synchronization model is a file that is created by the **Create Synchronization Model Wizard** in Sybase Central.

After you complete the **Create Synchronization Model Wizard**, you can continue to customize your model. No changes are made to your consolidated database or remote database until you deploy the model. Your model is stored in a model file with the extension *.mlsm*, and a reference to that file is stored in your synchronization model file..

When your model is complete, you use the **Deploy Synchronization Model Wizard** to deploy it. The **Deploy Synchronization Model Wizard** creates script files to run the MobiLink server and client using deployment options you choose. You can choose to make changes to your existing databases when you deploy or you can choose to have the wizard create files that you run yourself. Files that are created for the remote database can be used in remote tasks.

After you deploy, you can continue to customize the synchronization model or databases and then redeploy. If necessary, you can modify the deployed synchronization system out of Sybase Central, using the techniques that are described throughout the MobiLink documentation.

Creating synchronization models

Set up a MobiLink application with the Create Synchronization Model Wizard

1. Use the MobiLink plug-in to create a MobiLink project for your consolidated database if you have not already created one.

   See "Creating a MobiLink project" on page 22.

2. In the left pane of Sybase Central, expand your MobiLink project name, and then click **Synchronization Models**.

3. From the **File** menu, choose **New** » **Synchronization Model** to start the **Create Synchronization Model Wizard**.

4. From the **Welcome** page, choose a name for your synchronization model. Your model is stored as a *.mlsm* file in the project directory. Click **Next**.

5. On the **Primary Key Requirements** page, select the three checkboxes and then click **Next**. For more information about primary keys, see "Maintaining unique primary keys" [*MobiLink - Server Administration*].

6. On the **Consolidated Database Schema** page, select the consolidated database for obtaining your consolidated database schema from the list, and then click **Next**. the schema from the consolidated database is loaded.

   If you chose an Oracle database, you may be prompted to choose a subset of owners because loading schema for all owners can take a long time.

7. The **Remote Database Schema** page appears. You can create your remote database schema based on the consolidated database schema or an existing remote database. The existing remote database can be SQL Anywhere or UltraLite. For help deciding, see "Remote schemas" on page 25.

8. Follow the remaining instructions in the **Create Synchronization Model Wizard**. Default recommendations based on best practices are used where possible.

9. Click **Finish**.

   When you click **Finish** the synchronization model is opened in the left pane of Sybase Central. You are now working offline and you can make changes to the model. No changes are made outside the model until you deploy the model: the consolidated database does not change and the remote database is not created or changed until that time. See "Synchronization model tasks" on page 26 and "Deploying synchronization models" on page 38.

Remote schemas

A synchronization model contains schema for a remote database. This schema can be obtained from an existing remote database or from the consolidated database.

Use an existing remote database in the following cases:

- If you already have a remote database, especially if its schema is not a subset of the consolidated database schema.
- If your consolidated and remote columns need to have different types.
- If your remote tables need to have different owners from the tables on the consolidated database. For a new SQL Anywhere remote schemas created from the consolidate database, the owner of the remote tables is the same as the owner of the corresponding consolidated database tables. If you want a different owner, you should use an existing SQL Anywhere remote database with table ownership you set up.

Tip
You can manually change an existing database schema and then run the **Update Schema Wizard** to update the synchronization model in your MobiLink project. See "Updating schemas" on page 37.

When you deploy your model, you have three options for your remote database, regardless of how you created the remote schema in the model. Your deploy-time options for the remote database are:

- **Create a New Remote Database** Deployment can create a new remote database using the remote schema from the synchronization model. the database is created with default options.
- **Update an Existing Remote Database That Has No User Tables** If you deploy to an empty remote database, then the remote schema from the model is created in the database. This option is useful when you want to use non-default database creation options, such as collation.

  For SQL Anywhere databases, you can see a list of options that cannot be set after the database is created in the Remarks section of the initialization utility. See "Initialization utility (dbinit)" [*SQL Anywhere Server - Database Administration*].

  For UltraLite databases, database properties cannot be changed after the database is created. See "Choosing database creation parameters for UltraLite" [*UltraLite - Database Management and Reference*].
- **Update an Existing Remote Database That Has a Schema Matching the Schema in the Model** This option is useful when you have an existing remote database that you want to synchronize. When you deploy directly to an existing remote database, no changes are made to any existing remote data. If you try to deploy directly to an existing remote database whose schema doesn't match the remote schema in the model, you are prompted to update the remote schema in the model.

For a SQL Anywhere remote database, tables have the same owners as the original database. UltraLite database tables do not have owners.

See also

- "Deploying synchronization models" on page 38

Synchronization model tasks

You can perform several tasks with your synchronization model after creating it with the **Create Synchronization Model Wizard**. Changes are saved to the synchronization model file only. They are not saved to your consolidated or remote databases until the synchronization model is deployed.

You can modify synchronization models outside of Sybase Central but you cannot reverse-engineer changes back into the model. For example, you can add or modify MobiLink scripts using system procedures. See "MobiLink server system procedures" [*MobiLink - Server Administration*].

Modifying table and column mappings

Table mappings indicate which tables should be synchronized, how tables should be synchronized, and how the synchronized data should be mapped between the consolidated and remote databases.

Upload-only, download-only, and non-synchronized tables or columns

By default, MobiLink does a complete, bi-directional synchronization. You can change each table to be upload-only or download-only. You can also choose to not synchronize a table, which removes its table mapping.

In a synchronization model, you can only specify tables as download-only; you cannot create download-only publications.

To change the table mapping direction

1. In the left pane of Sybase Central, expand **MobiLink 12**, your MobiLink project name, **Synchronization Models**, and then select your synchronization model name.
2. Open the **Mappings** tab in the right pane.
3. In the **Table Mappings** pane, select a consolidated table.
4. In the **Mapping Direction** dropdown list, select one of the following:
   - **Not synchronized**. Choosing this option is the same as deleting the table mapping.
   - **Bi-directional**
   - **Download to Remote Only**
   - **Upload to Consolidated Only**

To remove a table mapping

1. In the left pane of Sybase Central, expand **MobiLink 12**, your MobiLink project name, **Synchronization Models**, and then select your synchronization model name.
2. Open the **Mappings** tab in the right pane.
3. In the **Table Mappings** pane, select a table mapping.
4. In the **Mapping Direction** dropdown list, choose **Not Synchronized**.

Changing table and column mappings

If your model is based on an existing remote database, the table and column mappings represent a best guess. You should check them and customize them as required.

To change table mappings

1. In the left pane of Sybase Central, expand **MobiLink 12**, your MobiLink project name, **Synchronization Models**, and then select your synchronization model name.
2. Open the **Mappings** tab in the right pane.
3. In the **Table Mappings** pane, select a table mapping.
4. To change the remote table that is mapped, from the **Remote Table** context menu, select a different table from the list of unsynchronized remote tables.
   - You can only choose remote tables that are not already mapped to consolidated tables.
   - If you want to add tables to your remote schema, see "Updating schemas" on page 37.
5. To add a table mapping for an unmapped consolidated table, use **File** » **New** » **Table Mappings** to open the **Create New Table Mappings** dialog where you choose the tables. Take care to not choose any shadow tables that were created from previously deploying synchronization models (for example, with the default suffixes of "_del" or "_mod").

   If you are creating a new remote schema from the consolidated schema, enable the option to also add the tables to the remote schema.

You can map a column in a synchronized consolidated table to a remote table column, a value determined when synchronizing, or exclude the column from synchronization. When mapping to a value, you can use the MobiLink user name, the remote database ID, or a SQL expression (which can include MobiLink named parameters). When you map a primary key column to a value and the table mapping is bi-directional, you need to prevent duplicate primary keys when downloading to remote databases.

To change a column mapping for a table mapping

1. In the **Table Mappings** pane, select the table mapping.
2. In the lower pane, open the **Column Mappings** tab.
3. Right-click the column mapping you want to change, and select one of the following from the context menu:
   - **None**
   - **MobiLink User Name**
   - **Remote ID**
   - **Custom**
   - An unmapped remote column

   To synchronize the consolidated column with a remote column, select the unmapped remote column from the bottom group of the menu. Only unmapped remote columns are listed.

To exclude the consolidated column from synchronization, choose **None**. The Direction icon shows that the consolidated column will not be synchronized.

To map the consolidated column to a value, you can choose the **MobiLink User Name**, the **Remote ID**, or use **Custom** to enter a SQL expression that is evaluated when the remote table's upload_insert, upload_update and upload_delete synchronization scripts are executed during the synchronization. The **Direction** icon shows that the value will only be uploaded; the consolidated column will not be downloaded to the remote database.

Modifying the download type

The download type of a table mapping can be timestamp, snapshot, or custom. You change the download type in the **Table Mappings** pane of the **Mappings** tab.

- **Timestamp-based download** Choose this option to use timestamp-based download as the default. Only rows that have been changed since the last synchronization are downloaded. See "Timestamp-based downloads" [*MobiLink - Server Administration*].
- **Snapshot download** Choose this option to use snapshot download as the default. All rows are downloaded even if they have not been changed since the last synchronization. See:
  - "Snapshot synchronization" [*MobiLink - Server Administration*]
  - "When to use snapshot synchronization" [*MobiLink - Server Administration*]
- **Custom download logic** Choose this option if you want to write your own download_cursor and download_delete_cursor scripts instead of having them generated for you. See:
  - "Writing synchronization scripts" [*MobiLink - Server Administration*]
  - "Writing download_cursor scripts" [*MobiLink - Server Administration*]
  - "Writing download_delete_cursor scripts" [*MobiLink - Server Administration*]

To change the download type

1. In the left pane of Sybase Central, expand **MobiLink 12**, your MobiLink project name, **Synchronization Models**, and then select your synchronization model name.
2. Open the **Mappings** tab in the right pane.
3. In the **Table Mappings** pane, select a table mapping.
4. In the **Download Type** dropdown list, select **Timestamp**, **Snapshot**, or **Custom**.
5. If you chose **Custom**, click the **Events** tab then type in your download_cursor script and download_delete_cursor scripts.

Modifying how deletes are recorded

If you are using snapshot download, all rows in the remote database are deleted before the snapshot is downloaded. If you are using timestamp-based download, you can decide how you want deletes on the consolidated database to be recorded for downloading to the remote database.

If you want to delete rows from remote databases when they are deleted from the consolidated database, you need to keep a record of the row to delete it. You can do this with shadow tables or by using logical deletes.

To change how deletes are recorded

1. In the left pane of Sybase Central, expand **MobiLink 12**, your MobiLink project name, **Synchronization Models**, and then select your synchronization model name.
2. Open the **Mappings** tab in the right pane.
3. In the **Table Mappings** pane, select a table mapping.
4. In the **Delete** column, select the checkbox if you want to download deletes from the consolidated database. Clear the checkbox if you do not want to download deletes from the consolidated database.
5. If you chose to download deletes, open the **Download Deletes** tab in the lower pane.

   To record deletions, you can choose to use a shadow table or logical deletes.

Logical deletes

The MobiLink synchronization model support for logical deletes assumes that a logical delete column is only on the consolidated database and not on the remote database. When copying a consolidated schema to a new remote schema, leave out any columns that match the logical delete column in the synchronization model settings.

Columns matching the default logical delete column name are automatically not copied to new remote schemas.

If you do want to use logical deletes in the remote database, choose to not download deletes and if necessary, update the remote schema to include the logical delete column.

> **Note**
> You need to set the column mapping for the logical delete column in the remote schema to the logical delete column in the consolidated schema.

See also

- “Handling deletes” [*MobiLink - Server Administration*]
- “Writing download_delete_cursor scripts” [*MobiLink - Server Administration*]
- “download_cursor table event” [*MobiLink - Server Administration*]

Modifying the download subset

Each MobiLink remote database can synchronize a subset of the data in the consolidated database. You can customize the download subset for each table.

The download subset options are:

- **User** Choose this option to partition data by MobiLink user name, which downloads different data to different registered MobiLink users.

  To use this option, the MobiLink user names must be in your consolidated database. You choose your MobiLink user names when you deploy, so you can choose names that match existing values on your consolidated database. (The column you use for MobiLink user names must be of a type that can hold the values you are using for the user name.) If the MobiLink user names are in a different table from the one you are subsetting, you must join to that table.

- **Remote ID** Choose this option to partition data by remote ID, which downloads different data to different remote databases.

  To use this option, the remote IDs must be in your consolidated database. Remote IDs are created as GUIDs by default, but you can set the remote IDs to match existing values on your consolidated database. (The column you use for remote IDs must be of a type that can hold the values you are using for the remote IDs.) If the remote IDs are in a different table from the one you are subsetting, you must join to that table.

  Note
  It is usually better to partition by user or by authentication parameter than by remote ID, because the remote ID can change if the remote computer is reset or replaced.

- **Custom** Choose this option to use a SQL expression that determines which rows are downloaded. Each synchronization only downloads rows where your SQL expression is true. This SQL expression is added to the WHERE clause of the generated download_cursor script. It is partly generated for you. You can use MobiLink named parameters in the expression. You can also refer to other tables. If you refer to other tables, you must list the other tables in the field above the expression, and include the join condition in your expression.

To change the download subset

1. In the left pane of Sybase Central, expand **MobiLink 12**, your MobiLink project name, **Synchronization Models**, and then select your synchronization model name.

2. Open the **Mappings** tab in the right pane.

3. In the **Table Mappings** pane, select a remote table.

4. In the **Download Subset** dropdown list, choose one of the following download subsets: **None**, **User**, **Remote**, or **Custom**.

5. If you chose **User**, **Remote**, or **Custom**, open the **Download Subset** tab in the lower pane.

6. If you chose **User** or **Remote**, the **Download Subset** tab allows you to identify the column that contains the MobiLink user names or remote IDs, either in the synchronized table or in a joined table. With a joined table, you must specify the columns for the join condition.

7. If you choose **Custom**, the **Download Subset** tab has two text boxes where you add information to construct a download_cursor script. You do not have to write a complete download_cursor. You only need to add extra information to identify the join and other restrictions on the download subset.

   - In the first text box (**Tables To Add To The Download Cursor's FROM Clause**), enter the table name(s) if your download_cursor requires a join to other tables. If the join requires multiple tables, separate them with commas.
   - In the second box (**SQL Expression To Use In The Download Cursor's WHERE Clause**), enter a SQL expression to be added to the generated WHERE clause that specifies the download subset condition and join condition. You can use MobiLink named parameters, including authentication parameters, in the expression.

See also

- "Introduction to MobiLink users" [*MobiLink - Client Administration*]
- "Remote IDs" [*MobiLink - Client Administration*]
- "Partitioning rows among remote databases" [*MobiLink - Server Administration*]
- "Using remote IDs and MobiLink user names in scripts" [*MobiLink - Client Administration*]
- "Using last download times in scripts" [*MobiLink - Server Administration*]
- "Writing download_cursor scripts" [*MobiLink - Server Administration*]

Example (User)

For example, the ULOrder table in CustDB can be shared between users. By default, orders are assigned to the employee who created them. But there are times when another employee needs to see orders created by someone else. For example, a manager may need to see all the orders created by employees in their department. The CustDB database has a provision for this via the ULEmpCust table. It allows you to assign customers to employees. They download all orders for that employee customer relationship.

To see how this is done, first view the download_cursor script for ULOrder without download subsetting. Select the ULEmpCust table in the **Mapping** tab. Choose Timestamp-based for the **Download Type** column and None for the **Download Subset** column. Right-click the table and choose **Go To Events**. The download_cursor for the table looks like this:

```
SELECT "DBA"."ULOrder"."order_id",
    "DBA"."ULOrder"."cust_id",
    "DBA"."ULOrder"."prod_id",
    "DBA"."ULOrder"."emp_id",
    "DBA"."ULOrder"."disc",
    "DBA"."ULOrder"."quant",
    "DBA"."ULOrder"."notes",
    "DBA"."ULOrder"."status"
FROM "DBA"."ULOrder"
WHERE "DBA"."ULOrder"."last_modified" >= {ml s.last_table_download}
```

Now go back to the **Mappings** tab. Change the **Download Subset** column for ULOrder to **User**. Open the **Download Subset** tab in the lower pane. Select **Use A Column In A Joined Relationship Table**. For the table to join, select ULEmpCust. For the column to match, select emp_id. The join condition should be emp_id = emp_id.

Right-click the table in the top pane and choose **Go To Events**. The download_cursor for the table now looks like this (the new lines are shown in bold):

```
SELECT "DBA"."ULOrder"."order_id",
    "DBA"."ULOrder"."cust_id",
    "DBA"."ULOrder"."prod_id",
    "DBA"."ULOrder"."emp_id",
    "DBA"."ULOrder"."disc",
    "DBA"."ULOrder"."quant",
    "DBA"."ULOrder"."notes",
    "DBA"."ULOrder"."status"
FROM "DBA"."ULOrder", "DBA"."ULEmpCust"
WHERE "DBA"."ULOrder"."last_modified" >= {ml s.last_table_download}
AND "DBA"."ULOrder"."emp_id" = "DBA"."ULEmpCust"."emp_id"
AND "DBA"."ULEmpCust"."emp_id" = {ml s.username}
```

Example (Custom)

For example, assume you want to subset the download of a table called Customer by MobiLink user and you also want to only download rows where active=1. The MobiLink user names do not exist in the table you are subsetting, so you need to create a join to a table called SalesRep, which contains the MobiLink user names.

In the **Mappings** tab, Choose **Timestamp-based** for the **Download Type** column and **Custom** for the **Download Subset** column of the Customer table mapping. Open the **Download Subset** tab in the lower pane. In the first box (**Tables To Add To The Download Cursor's FROM Clause**), type:

```
SalesRep
```

In the second box (**SQL Expression To Use In The Download Cursor's WHERE Clause**), type:

```
SalesRep.ml_username = {ml s.username}
   AND Customer.active = 1
   AND Customer.cust_id = SalesRep.cust_id
```

Right-click the table in the top pane and choose **Go To Events**. The download_cursor for the table now looks like this:

```
SELECT "DBA"."Customer"."cust_id",
 "DBA"."Customer"."cust_name"
FROM "DBA"."Customer", SalesRep
WHERE "DBA"."Customer"."last_modified" >= {ml s.last_table_download}
 AND SalesRep.ml_username = {ml s.username}
 AND Customer.active = 1
 AND Customer.cust_id = SalesRep.cust_id
```

The final line of the WHERE clause creates a key join of Customer to SalesRep.

Download delete subset

If you are using a download subset to synchronize a subset of the data on the consolidated database, then by default the download delete subset is set to **Same**, which makes it exactly the same as the download subset. You can choose to change it to **None** or **Custom**. For more information about download subsets, see “Modifying the download subset” on page 30.

Modifying conflict detection and resolution

When a row is updated on both the remote and consolidated databases, a conflict occurs the next time the databases are synchronized.

You have the following options for detecting conflicts:

- **No conflict detection** Choose this option if you do not want any conflict detection. Uploaded updates are applied without checking for conflicts. This avoids having to fetch current row values from the consolidated database, so the synchronization of updates may be faster.
- **Row-based conflict detection** A conflict is detected if the row has been updated by both the remote and consolidated databases since the last synchronization.

  This option defines an upload_fetch script and upload_update script. See "Detecting conflicts with upload_fetch or upload_fetch_column_conflict scripts" [*MobiLink - Server Administration*].
- **Column-based conflict detection** A conflict is detected if the same column has been updated for the row in both the remote and consolidated databases.

  This option defines an upload_fetch_column_conflict script. See "Detecting conflicts with upload_fetch or upload_fetch_column_conflict scripts" [*MobiLink - Server Administration*].

  If a table has BLOBs and you choose column-based conflict detection, row-based conflict detection is used.

You have the following options for resolving conflicts:

- **Consolidated** First in wins: uploaded updates that conflict are rejected.
- **Remote** Last in wins: uploaded updates are always applied.

  Only use this option with column-based conflict detection. Otherwise, you get the same results and better performance by choosing no conflict detection.
- **Timestamp** The newest update wins. To use this option, you must create and maintain a timestamp column for the table. This timestamp column should record the last time that a row was changed. The column should exist on both the consolidated and remote databases and not be the same column used for timestamp-based downloads. To work, your remote and consolidated databases must use the same time zone (preferably UTC) and their clocks must be synchronized.
- **Custom** You write your own resolve_conflict scripts. You do this in the **Events** tab after the wizard completes.

  See "Resolving conflicts with resolve_conflict scripts" [*MobiLink - Server Administration*].

To customize conflict detection and resolution

1. In the left pane of Sybase Central, expand **MobiLink 12**, your MobiLink project name, **Synchronization Models**, and then select your synchronization model name.

2. Open the **Mappings** tab in the right pane.
3. In the **Table Mappings** pane, select a table mapping.
4. In the **Conflict Detection** dropdown list, choose **None**, **Row-based**, or **Column-based**. If you chose **None**, you are done.
5. If you chose **Row-based** or **Column-based**, choose **Consolidated**, **Remote**, **Timestamp**, or **Custom** from the **Conflict Resolution** dropdown list.
6. If you chose **Timestamp** conflict resolution, open the **Conflict Resolution** tab in the lower pane and enter the name of a timestamp column to use.
7. If you chose **Custom** conflict resolution, open the **Events** tab and write a resolve_conflict script for the table.

See also

- "Handling conflicts" [*MobiLink - Server Administration*]

Modifying scripts in a synchronization model

You modify scripts in a synchronization model using the **Events** tab.

The **Events** tab allows you to perform the following tasks:

- View and modify the scripts that were generated by the **Create Synchronization Model Wizard**.
- Create new scripts.

The top of the **Events** tab indicates the group that the selected script belongs to. All scripts for a single table are grouped together. The top of the **Events** tab also indicates the name of the selected script and whether it was generated by the **Create Synchronization Model Wizard**, whether it was user-defined, or whether a generated script was overridden. It also indicates whether the synchronization logic is written in SQL, .NET, or Java.

The script is fully under your control when you add a script of override a generated script; it does not change automatically when you change a related setting. For example, if you change a download_delete_cursor for a model and then clear the **Del** column in the **Table Mappings** pane under the **Mappings** tab, your customized download_delete_cursor is not affected.

You can use options in the **File** menu to restore generated scripts you have overridden, restore scripts that you have set to be ignored, or to remove new scripts you have added. Select the script(s) you want to restore or remove and choose **File** to view your options.

To locate a script for a particular table

1. In the left pane of Sybase Central, expand **MobiLink 12**, your MobiLink project name, **Synchronization Models**, and then select your synchronization model name.

2. Open the **Table Mappings** tab in the right pane.
3. In the **Table Mappings** pane, select a table mapping.
4. From the **File** menu, choose **Go To Events**.

   The **Events** tab opens, and you can view all scripts for that particular table.
5. In the **Event** field, choose the script name that you want to locate.

   The cursor moves to that script and you can make the necessary changes.

   Existing scripts are highlighted in bold.

Authenticating to an external server

You enable synchronization authentication to an external server for a synchronization model using the **Authentication** tab.

To authenticate to an external POP3, IMAP, or LDAP server

1. In the left pane of Sybase Central, expand **MobiLink 12**, your MobiLink project name, **Synchronization Models**, and then select your synchronization model name.
2. Open the **Authentication** tab in the right pane.
3. Select **Enable Custom Authentication For This Synchronization Model**.
4. Select the server that you want to authenticate to.
5. Enter the appropriate host, port, and URL information in their respective fields.

   For more information about these fields, see "External authenticator properties" [*MobiLink - Client Administration*].

Setting up server-initiated synchronization in a synchronization model

Server-initiated synchronization allows you to initiate synchronization on the client database when something changes on the consolidated database. You can enable server-initiated synchronization in a synchronization model. This method allows you to deploy a limited version of server-initiated synchronization that is easy to set up and run.

When enabling server-initiated synchronization in a synchronization model, the MobiLink server uses a download_cursor script for a table to determine when to initiate a synchronization. It uses the download_cursor script to generate a request_cursor for the Notifier. You cannot customize your request_cursor using this method.

To set up server-initiated synchronization in a synchronization model

1. In the left pane of Sybase Central, expand **MobiLink 12**, your MobiLink project name, **Synchronization Models**, and then select your synchronization model name.
2. Open the **Notification** tab in the right pane.
3. Select **Enable Server-Initiated Synchronization**.
4. Select a consolidated database table to use for notification.

   A change to the data in this table results in a notification being sent to the remote database. The notification triggers a synchronization.

   Note
   You can only choose tables for this purpose for which you have defined a timestamp-based download cursor (the default). The notification is based on the contents of the download cursor.

5. Select a polling interval.

   The polling interval is the time between polls. You can choose a predefined polling interval or you can specify your own. The default is 30 seconds.

   If the Notifier loses the database connection, it recovers automatically at the first polling interval after the database becomes available again.

6. Change the isolation level of the Notifier's database connection. (Optional)

   The default is read committed. Be aware of the consequences of setting the isolation level. Higher levels increase contention, and may adversely affect performance. Isolation level 0 allows reads of uncommitted data—data which may eventually be rolled back.

See also

- "MobiLink - Server-Initiated Synchronization"
- "Quick start guide to server-initiated synchronization" [*MobiLink - Server-Initiated Synchronization*]

Updating schemas

The **Update Schema Wizard** allows you to update the consolidated and remote database schemas in your synchronization model.

The **Update Schema Wizard** is most useful after you have deployed your model and:

- You made a change to the remote database schema that needs to be included in the model.
- You made a change to the consolidated database schema that needs to be included in the model.

For example, you need to run Update Schema before redeploying a model that created timestamp-based download for one or more tables. The previous deployment changed the schema of the consolidated database by adding a timestamp column or shadow table, so the schema needs to be updated.

To update the schema of a synchronization model

1. In the left pane of Sybase Central, expand **MobiLink 12**, your MobiLink project name, **Synchronization Models**, and then select your synchronization model name.
2. Choose **File** » **Update Schema**.
3. Choose one of the following options:
   - **The Consolidated Database Schema** The consolidated schema in the model is updated. The remote schema in the model is unchanged.
   - **The Remote Database Schema** The remote schema in the model is updated. The consolidated schema in the model is unchanged.
   - **Both The Consolidated And Remote Database Schemas** Both the consolidated and remote schemas are updated in the model to match the schemas of the existing databases.
4. Follow the instructions in the **Update Schema Wizard**.

   When you click **Finish**, no changes are made outside the model until you deploy the synchronization model; the consolidated database does not change and the remote database is not created or changed until that time.
5. Map the new remote tables in the **Mappings** tab. See "Modifying table and column mappings" on page 27.

Deploying synchronization models

You deploy synchronization models with the **Deploy Synchronization Model Wizard**.

The following items can be deployed:

- Changes to the consolidated database.
- SQL Anywhere or UltraLite remote databases (you can choose to create a database, or add tables to an existing empty database, or use an existing database that already has your remote tables).
- Batch files to deploy the model (the generated batch files have variable declarations at the beginning that you can edit before running the batch files).
- Batch files to run the MobiLink server and the MobiLink client.
- Server-initiated synchronization configuration.

Deploying to the consolidated database

The **Deploy Synchronization Model Wizard** provides two options for deploying to the consolidated database:

- Apply your synchronization model directly to your consolidated database by populating MobiLink system tables and creating all required shadow tables, columns, triggers, and stored procedures.
- Create a UTF-8 encoded SQL file that contains all the same changes and a batch file to run the SQL file against your consolidated database. You can inspect this file, alter it, and run it anytime. The effect is identical to applying the changes directly.

To deploy a consolidated database from a SQL file

- When you ran the **Deploy Synchronization Model Wizard**, if you chose to create a file to run later (on the **Consolidated Database Deployment Destination** page), you must run the batch file that is located in the *consolidated* sub-folder of your model. This file creates all the objects you chose to have created in the consolidated database, including synchronization scripts, shadow tables, and triggers. It can also register MobiLink users in the consolidated database.

  To run this file, navigate to the *consolidated* directory and run the file that ends with *_consolidated.bat* or *_consolidated.sh*. You must include connection information. For example, run:

  ```
  MyModel_consolidated.bat
  "dsn=my_odbc_datasource;uid=myuserid;pwd=mypassword"
  ```

  For some drivers, the ODBC data source can include the user ID and password so they do not need to be specified.

Note
If your deployment creates shadow tables, you must connect to the consolidated database as either the owner of the base tables for which shadow tables are created, or as an administrator.

Deploying remote databases

You can choose to use an existing remote database or have the wizard create one for you. The wizard can create remote databases directly or you can have it create a UTF-8 encoded SQL file and a batch file that you run to create or update remote databases.

The wizard creates a remote database (either SQL Anywhere or UltraLite) with default database creation options using the database owner that you specified in the model. Alternatively, you can create a remote database outside the **Deploy Synchronization Model Wizard** with your own custom settings and use the wizard to add the required remote tables, or you can deploy to an existing remote database that already has the remote tables.

To deploy a remote database from a SQL file

- When you ran the **Deploy Synchronization Model Wizard**, if you chose to create a file to run later (on the **New SQL Anywhere Remote Database** page or **New UltraLite Remote Database** page), you must run the batch file that was created with the SQL file in the *remote* directory. This file creates all the objects you chose to have created in the remote database, including tables, publications, subscriptions, and MobiLink users.

To run this file, navigate to the *remote* directory and run the file that ends with *_remote.bat* or *_remote.sh*. For example, run:

```
MyModel_remote.bat
```

You are prompted for a password if you are using an existing remote database.

Deploying batch files to run synchronization tools

The wizard can create the following batch files:

- A batch file to run the MobiLink server with options that you specify.
- For SQL Anywhere remote databases, a batch file to run dbmlsync with options that you specify.
- For UltraLite remote databases, a batch file to run ulsync with options that you specify. Ulsync is used for testing synchronization, so it helps you get started when you don't have a working UltraLite application.
- If you set up server-initiated synchronization for your synchronization model, the **Deploy Synchronization Model Wizard** can also create batch files to run the Notifier and the Listener.

To deploy a model

1. In the left pane of Sybase Central, expand **MobiLink 12**, your MobiLink project name, **Synchronization Models**, and then select your synchronization model name.
2. From the **File** menu, choose **Deploy**.
3. Follow the instructions in the **Deploy Synchronization Model Wizard**.
4. When you are finished, the changes you selected are deployed. If there are existing files of the same name, they are overwritten.
5. To synchronize your application, see "Synchronizing a deployed model" on page 41.

Redeploying a model

You can alter a synchronization model after deploying it by making changes to the synchronization model and then redeploying. You can also alter your model using system procedures or other methods. However, you cannot reverse-engineer the changes back into the synchronization model when you alter a deployed model outside of Sybase Central. Changes made outside of Sybase Central are overwritten when you redeploy the model.

Deployment often causes schema changes, so you may need to update the schema even if you haven't made any other changes. For example, if you deploy a model that adds a timestamp column to each synchronized table on the consolidated database (which is the default behavior when you create a model), you need to update the consolidated schema in the model before redeploying. Likewise, if you add a table to the consolidated database and then want to redeploy, you need to update the consolidated schema in the model and then create new remote tables.

See "Updating schemas" on page 37.

Note
Redeploying a synchronization model drops the recreates shadow tables. To avoid losing shadow table data, deploy the file and edit the SQL file to not recreate the shadow tables.

Synchronizing a deployed model

When you deploy a model, directories and files are optionally created under the location you chose on the first page of the **Create Synchronization Model Wizard**. The files and directories are named according to the model name you chose at that time.

Assume you named your model **MyModel** and saved it under *c:\SyncModels*. Depending on the deployment options you chose, you might have the following files:

Directories (based on example name and location)	Description and contents (based on example name)
c:\SyncModels	Contains your model file, saved as *MyModel.mlsm*.
c:\SyncModels\MyModel	Contains folders holding your deployment files.
c:\SyncModels\MyModel\consolidated	Contains deployment files for the consolidated database: • *MyModel_consolidated.sql* - a SQL file for setting up the consolidated database. • *MyModel_consolidated.bat* - a batch file for running the SQL file.
c:\SyncModels\MyModel\mlsrv	Contains deployment files for the MobiLink server: • *MyModel_mlsrv.bat* - a batch file for running the MobiLink server. If you have set up server-initiated synchronization, it also starts the Notifier.

Directories (based on example name and location)	Description and contents (based on example name)
c:\SyncModels\MyModel\remote	Contains deployment files for the remote databases: • *dblsn.txt* - if you set up server-initiated synchronization, this is a text file with Listener option settings. It is used by *MyModel_dblsn.bat*. • *MyModel_dblsn.bat* - if you set up server-initiated synchronization, this is a batch file for running the Listener. • *MyModel_dbmlsync.bat* - if you deployed a SQL Anywhere remote database, this is a batch file for synchronizing SQL Anywhere databases with dbmlsync. • *MyModel_remote.bat* - a batch file for running *MyModel_remote.sql*. • *MyModel_remote.db* - if you chose to create a new SQL Anywhere remote database, this is the database file. • *MyModel_remote.sql* - a SQL file for setting up the new SQL Anywhere remote database. • *MyModel_remote.udb* - if you chose to create a new UltraLite remote database, this is the database file. • *MyModel_ulsync.bat* - if you deployed an UltraLite database, a batch file for testing synchronization with an UltraLite remote database using the ulsync utility.

Running the batch files

You must run the batch files that are created by the **Deploy Synchronization Model Wizard** from the command line, and for many you must include connection information. You may need to create ODBC data sources before running these batch files.

See "Creating ODBC data sources" [*SQL Anywhere Server - Database Administration*].

To synchronize your synchronization model using batch files

1. If you have not yet run MobiLink setup scripts on consolidated database, run them before deploying.

   See "Setting up a consolidated database" [*MobiLink - Server Administration*].

2. When you ran the **Deploy Synchronization Model Wizard**, if you chose to create a file to run later (on the **Consolidated Database Deployment Destination** page), you must run the batch file that is located in the *consolidated* sub-folder of your model. This file creates all the objects you chose to have created in the consolidated database, including synchronization scripts, shadow tables, and triggers. It can also register MobiLink users in the consolidated database.

   To run this file, navigate to the *consolidated* directory and run the file that ends with *_consolidated.bat* or *_consolidated.sh*. You must include connection information on the command line. For example, run:


```
MyModel_consolidated.bat "dsn=MY_ODBC_DATASOURCE"
```

3. When you ran the **Deploy Synchronization Model Wizard**, if you chose to create a file to run later (on the **New SQL Anywhere Remote Database** page or **New UltraLite Remote Database** page), you must run the batch file in the *remote* directory. This file creates all the objects you chose to have created in the remote database, including tables, publications, subscriptions, and MobiLink users.

   To run this file, navigate to the *remote* directory and run the file that ends with *_remote.bat* or *_remote.sh*. For example, run:

   ```
   MyModel_remote.bat
   ```

   You are prompted for a password if you are using an existing remote database.

4. Start the MobiLink server by running *mlsrv\MyModel_mlsrv.bat*. If you set up server-initiated synchronization, this also starts the Notifier. You must include connection information for the consolidated database on the command line. For example, run:

   ```
   MyModel_mlsrv.bat "dsn=MY_ODBC_DATASOURCE"
   ```

5. Synchronize.

   For a SQL Anywhere remote database:

   - Grant REMOTE DBA authority to a user other than DBA (recommended). For example, execute the following in Interactive SQL:

     ```
     GRANT REMOTE DBA
     TO userid, IDENTIFIED BY password
     ```

   - Connect as the user with REMOTE DBA authority.
   - Start the remote database that is located in the *remote* directory. For example, run:

     ```
     dbeng12 MyModel_remote.db
     ```

   - Start dbmlsync, the SQL Anywhere MobiLink client. Run the file that ends with *_dbmlsync.bat* in the *remote* directory. You must include connection information on the command line. For example, run:

     ```
     MyModel_dbmlsync.bat "uid=dba;pwd=sql;server=MyModel_remote"
     ```

   For an UltraLite remote database:

   - To test your synchronization, run the file that ends with *_ulsync.bat* in the *remote* directory.
   - Alternatively, run your UltraLite application.

6. If you set up server-initiated synchronization, you need to perform a first synchronization and then start the Listener. The first synchronization is required to create a remote ID file. To start the Listener, run the file that ends with *_dblsn.bat* in the *remote* directory. For example, run:

   ```
   MyModel_dblsn.bat
   ```

See also

- "GRANT REMOTE DBA statement [MobiLink] [SQL Remote]" [*SQL Anywhere Server - SQL Reference*]
- "Permissions for dbmlsync" [*MobiLink - Client Administration*]

Limitations of synchronization models

The following are some restrictions of synchronization models:

- **Changes made outside the model cannot be redeployed** If you deploy a synchronization model and then make changes to it outside the model, those changes are not saved in the model. This practice is fine if you want to use the model as a starting point, deploy, and then make all your changes outside the model. However, if you want to redeploy the model, you are better off making your changes to your MobiLink project so that they are saved and can be redeployed.

- **Versions** A synchronization model can have only one version. See "Script versions" [*MobiLink - Server Administration*].

- **MobiLink system database** You cannot use a MobiLink system database that is separate from the consolidated database when deploying a synchronization model. See "MobiLink system database" [*MobiLink - Server Administration*].

- **Multiple publications** You cannot create multiple publications. After you have deployed your model you can add more publications using non-model methods such as the CREATE PUBLICATION statement, but you cannot reverse-engineer these additions back into your model. See "Publishing data" [*MobiLink - Client Administration*].

- **Views** It is not possible to select a view when you are selecting consolidated database tables for table mappings.

- **Computed columns** You cannot upload to computed columns in a consolidated database table. If you deploy a synchronization model with computed columns, the deployment may have errors creating the trigger used for timestamp-based downloads. You can either exclude the column from synchronization, or configure the table as download-only (and either use snapshot download or edit the generated consolidated SQL file to remove the computed column from the trigger definition).

  Copying computed columns causes a syntax error when deploying the new remote schema to create a new remote database. When dealing with computed columns you should do one of the following:

  - Deploy the synchronization model to an existing remote database.

  - Exclude the computed column from the remote schema. Note that if you want to synchronize a consolidated database table that has computed columns, you cannot upload to the table.

  The Microsoft SQL Server AdventureWorks sample database contains computed columns. Set the columns to be download-only or exclude the columns from synchronization when using this database to create a model.

Deployment considerations

- **Spatial columns** Spatial columns are copied, though the spatial subtype and SRID may not be copied if the consolidated RDBMS does not have meta data support for obtaining those, such as the ST_GEOMETRY_COLUMNS view of the SQL/MM standard. Spatial support in UltraLite is limited to one type (ST_GEOMETRY) that only supports point values and column SRID constraints of SRID=0 or SRID=4326, so you may get a warning or error when deploying an incompatible spatial type to a new UltraLite database.

- **Long object names** The database objects that are created when deploying may have names that are longer than the database supports (because the new object names are created by adding suffixes to the base table names). If this happens, deploy only to file (not directly to a database) and edit the generated SQL file to replace all occurrences of the name that is too long.

- **New remote schemas** If you create a new remote schema in the **Create Synchronization Model Wizard**, the new remote database columns do not contain indexes of the columns in the consolidated database. Foreign keys and default column values are copied to the new remote database, however, this support relies on database meta data returned by the ODBC driver and syntax or other errors may occur due to driver problems. For example, if a driver reports a default column value in a format that cannot be used to declare such a default in a SQL Anywhere or UltraLite remote database, then errors can occur (including syntax errors when deploying).

  UltraLite does not support NCHAR(n), NVARCHAR(n), LONG NVARCHAR, VARBIT, or LONG VARBIT column types. When deploying a synchronization model to a new UltraLite database, such columns in the remote schema are converted to CHAR(4n), VARCHAR(4n), or LONG VARCHAR. If 4n is larger than the maximum length for CHAR and VARCHAR, the maximum length is used and you get a warning.

  You can use an existing remote database to create a synchronization model or to update the remote schema in a model.

- **Proxy tables** It is possible to synchronize with consolidated database tables that are proxy tables to another database, but you need to add the TIMESTAMP column to both the base table and the proxy table if you use a TIMESTAMP column for timestamp-based downloads. The **Deploy Synchronization Model Wizard** cannot add a column to a proxy table or its base, so you either need to use an existing column on both the base and proxy, or you need to use a shadow table or snapshot download.

- **Materialized views** If you are using timestamp-based downloads and have chosen to add a timestamp column to consolidated tables, you must disable any materialized views that depend on the tables before deploying. Otherwise you may get errors when trying to alter the tables. For SQL Anywhere consolidated databases, use the sa_dependent_views system procedure to find out if a table has dependent materialized views. See "sa_dependent_views system procedure" [*SQL Anywhere Server - SQL Reference*].

Other considerations

- **Creating remote databases based on an Oracle consolidated database** When you are using an Oracle consolidated database as the basis for your SQL Anywhere or UltraLite remote database,

you may want to change DATE columns in the consolidated database to TIMESTAMP. Otherwise, sub-second information is lost on upload.

Exploring the CustDB sample for MobiLink

Introduction to the MobiLink CustDB tutorial

CustDB is a sales-status application. The CustDB sample is a valuable resource for the MobiLink developer. It provides you with examples of how to implement many of the techniques you need to develop MobiLink applications.

The application has been designed to illustrate several common synchronization techniques. To get the most out of this section, study the sample application as you read.

A version of CustDB is supplied for each supported operating system and for each supported database type.

For the locations of CustDB and setup instructions, see "Setting up the CustDB consolidated database" on page 47.

CustDB Scenario

A consolidated database is located at the head office. The following data is stored in the consolidated database:

- The MobiLink system tables that hold the synchronization metadata, including the synchronization scripts that implement synchronization logic.
- The CustDB data, including all customer, product, and order information, stored in the rows of base tables.

There are two types of remote databases, mobile managers and sales representatives.

Each mobile sales representative's database contains all products but only those orders assigned to that sales representative, while a mobile manager's database contains all products and orders.

Synchronization design

The synchronization design in the CustDB sample application uses the following features:

- **Complete table downloads** All rows and columns of the ULProduct table are shared in their entirety with the remote databases.
- **Column subsets** All rows, but not all columns, of the ULCustomer table are shared with the remote databases.
- **Row subsets** Different remote users get different sets of rows from the ULOrder table.

For more information about row subsets, see "Partitioning rows among remote databases" [*MobiLink - Server Administration*].

- **Timestamp-based synchronization** This is a way of identifying changes that were made to the consolidated database since the last time a device synchronized. The ULCustomer and ULOrder tables are synchronized using a method based on timestamps.

  See "Timestamp-based downloads" [*MobiLink - Server Administration*].

- **Snapshot synchronization** This is a simple method of synchronization that downloads all rows in every synchronization. The ULProduct table is synchronized in this way.

  See "Snapshot synchronization" [*MobiLink - Server Administration*].

- **Primary key pools to maintain unique primary keys** It is essential to ensure that primary key values are unique across a complete MobiLink installation. The primary key pool method used in this application is one way of ensuring unique primary keys.

  See "Using primary key pools" [*MobiLink - Server Administration*].

  For other ways to ensure that primary keys are unique, see "Maintaining unique primary keys" [*MobiLink - Server Administration*].

For an entity-relationship diagram of the CustDB tables, see "The CustDB sample database application" [*SQL Anywhere 12 - Introduction*].

Further reading

- "UltraLite CustDB samples" [*UltraLite - Database Management and Reference*]

CustDB setup

This section describes the pieces that make up the code for the CustDB sample application and database. These include:

- The sample SQL scripts, located in the *samples-dir\MobiLink\CustDB*.
- The application code, located in *samples-dir\UltraLite\CustDB*.
- Platform-specific user interface code, located in subdirectories of *samples-dir\UltraLite\CustDB* named for each operating system.

Note
For more information about *samples-dir*, see "Samples directory" [*SQL Anywhere Server - Database Administration*].

Setting up the CustDB consolidated database

The CustDB consolidated database can be any MobiLink supported consolidated database.

SQL Anywhere 12 CustDB

A SQL Anywhere 12 CustDB consolidated database is provided in *samples-dir\UltraLite\CustDB\custdb.db*. A DSN called SQL Anywhere 12 CustDB is included with your installation.

You can rebuild this database using the file *samples-dir\UltraLite\CustDB\makedbs.cmd*.

If you want to explore the way the CustDB sample is created, you can view the file *samples-dir\MobiLink\CustDB\syncsa.sql*.

CustDB for other RDBMSs

The following SQL scripts are provided in *samples-dir\MobiLink\CustDB* to build the CustDB consolidated database as any one of these supported RDBMSs:

RDBMS	Custdb setup script
Adaptive Server Enterprise	*custase.sql*
SQL Server	*custmss.sql*
Oracle	*custora.sql*
DB2 LUW	*custdb2.sql*
MySQL	*custmys.sql*

The following procedures create a CustDB consolidated database for each of the supported RDBMS.

For more information about preparing a database for use as a consolidated database, see "Setting up a consolidated database" [*MobiLink - Server Administration*].

To set up a consolidated database (Adaptive Server Enterprise, MySQL, Oracle, SQL Server)

1. Create a database in your RDBMS.
2. Add the MobiLink system objects by running one of the following SQL scripts, located in the *MobiLink\setup* subdirectory of your SQL Anywhere 12 installation:
   - For an Adaptive Server Enterprise consolidated database, run *syncase.sql*.
   - For a MySQL consolidated database, run *syncmys.sql*.
   - For an Oracle consolidated database, run *syncora.sql*.
   - For a SQL Server consolidated database, run *syncmss.sql*.
3. Add sample user tables, stored procedures and MobiLink synchronization scripts to the CustDB database by running one of the following SQL scripts, located in *samples-dir\MobiLink\CustDB*:
   - For an Adaptive Server Enterprise consolidated database, run *custase.sql*.

- For a MySQL consolidated database, run *custmys.sql*.
- For an Oracle consolidated database, run *custora.sql*.
- For a SQL Server consolidated database, run *custmss.sql*.

4. Create an ODBC data source called CustDB that references your database on the client computer.
   a. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **ODBC Data Source Administrator**.
   b. Click **Add**.
   c. Select the appropriate driver from the list.

      Click **Finish**.
   d. Name the ODBC data source CustDB.
   e. Click the **Login** tab. Enter the **User ID** and **Password** for your database.

To set up a consolidated database (DB2 LUW)

1. Create a consolidated database on the DB2 LUW server. For the purposes of this tutorial, name it CustDB.
2. Ensure that the default table space (usually called USERSPACE1) uses 8 KB pages.

   If the default table space does not use 8 KB pages, complete the following steps:
   a. Verify that at least one of your buffer pools has 8 KB pages. If not, create a buffer pool with 8 KB pages.
   b. Create a new table space and temporary table space with 8 KB pages.

      For more information, consult your DB2 LUW documentation.
3. Add the MobiLink system objects to the DB2 LUW consolidated database using the file *MobiLink\setup\syncdb2.sql*:
   a. Change the connect command at the top of the file *syncdb2.sql*. Replace *DB2Database* with the name of your database (or its alias). In this example, the database is called CustDB. You can also add your DB2 user name and password as follows:

      ```
      connect to CustDB user userid using password ~
      ```
   b. Open a DB2 LUW Command Window on either the server or client computer. Run *syncdb2.sql* by typing the following command:

      ```
      db2 -c -ec -td~ +s -v -f syncdb2.sql
      ```
4. Add data tables, stored procedures and MobiLink synchronization scripts to the CustDB database:
   a. If necessary, change the connect command in custdb2.sql. For example, you could add the user name and password as follows. Replace *userid* and *password* with your user name and password.

      ```
      connect to CustDB user userid using password
      ```
   b. Open a DB2 Command Window on either the server or client computer.
   c. Run custdb2.sql by typing the following command:

```
db2 -c -ec -td~ +s -v -f custdb2.sql
```

d. When processing is complete, enter the following command to close the command window:

```
exit
```

5. Create an ODBC data source called CustDB that references the DB2 LUW database on the DB2 LUW client.

   a. Start the ODBC Data Source Administrator:

   Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **ODBC Data Source Administrator**.

   The ODBC Data Source Administrator appears.

   b. On the **User DSN** tab, click **Add**.

   c. In the **Create New Data Source** window, select the ODBC driver for your DB2 LUW database. For example, choose IBM DB2 UDB ODBC Driver. Click **Finish**.

   For information about how to configure your ODBC driver, see:

   - Your DB2 LUW documentation
   - http://www.sybase.com/detail?id=1011880

See also

- "IBM DB2 LUW consolidated database" [*MobiLink - Server Administration*]

Setting up an UltraLite remote database

The following procedure creates a remote database for CustDB. The CustDB remote database must be an UltraLite database.

The application logic for the remote database is located in *samples-dir\UltraLite\CustDB*. It includes the following files:

- **Embedded SQL logic** The file *custdb.sqc* contains the SQL statements needed to query and modify information from the UltraLite database, and the calls required to start synchronization with the consolidated database.
- **C++ API logic** The file *custdbcomp.cpp* contains the C++ API logic.
- **User-interface features** These features are stored separately, in platform-specific subdirectories of *Samples\UltraLite\CustDB*.

You complete the following steps to install the sample application to a remote device that is running UltraLite:

To install the sample application to a remote device

1. Start the consolidated database.
2. Start the MobiLink server.

3. Install and start the sample application to your client device.

4. Synchronize the sample application.

Example

The following example installs the CustDB sample on a Windows desktop running against a DB2 consolidated database.

1. Ensure that the consolidated database is running:

   For a DB2 LUW database, open a DB2 Command Window. Type the following command, where *userid* and *password* are the user ID and password for connecting to the DB2 LUW database:

   ```
   db2 connect to CustDB user userid using password
   ```

2. Start the MobiLink server:

   For a DB2 LUW database, at a command prompt, run the following command:

   ```
   mlsrv12 -c "DSN=CustDB" -zp
   ```

3. Start the CustDB sample application:

   a. From the **Start** menu, choose **Programs** » **SQL Anywhere 12** » **MobiLink** » **Synchronization Server Sample**.

   b. Enter a value of 50 for the employee ID and click **OK**.

      The application automatically synchronizes and a set of customers, products, and orders are downloaded to the application from the CustDB consolidated database.

4. Synchronize the remote application with the consolidated database.

   From the **File** menu, choose **Synchronize**.

   You only need to complete this step when you have made changes to the database.

Tables in the CustDB databases

The table definitions for the CustDB database are in platform-specific files in *samples-dir\MobiLink\CustDB*. (For information about *samples-dir*, see "Samples directory" [*SQL Anywhere Server - Database Administration*].)

For an entity-relationship diagram of the CustDB tables, see "The CustDB sample database application" [*SQL Anywhere 12 - Introduction*].

Both the consolidated and the remote databases contain the following five tables, although their definitions are slightly different in each location.

ULCustomer

The ULCustomer table contains a list of customers.

In the remote database, ULCustomer has the following columns:

- **cust_id** A primary key column that holds a unique integer that identifies the customer.
- **cust_name** A 30-character string containing the name of the customer.

In the consolidated database, ULCustomer has the following additional column:

- **last_modified** A timestamp containing the last time the row was modified. This column is used for timestamp-based synchronization.

ULProduct

The ULProduct table contains a list of products.

In both the remote and consolidated databases, ULProduct has the following columns:

- **prod_id** A primary key column that contains a unique integer that identifies the product.
- **price** An integer identifying the unit price.
- **prod_name** A 30-character string that contains the name of the product.

ULOrder

The ULOrder table contains a list of orders, including details of the customer who placed the order, the employee who took the order, and the product being ordered.

In the remote database, ULOrder has the following columns:

- **order_id** A primary key column that holds a unique integer identifying the order.
- **cust_id** A foreign key column referencing ULCustomer.
- **prod_id** A foreign key column referencing ULProduct.
- **emp_id** A foreign key column referencing ULEmployee.
- **disc** An integer containing the discount applied to the order.
- **quant** An integer containing the number of products ordered.
- **notes** A 50-character string containing notes about the order.
- **status** A 20-character string describing the status of the order.

In the consolidated database, ULOrder has the following additional column:

- **last_modified** A timestamp containing the last time the row was modified. This column is used for timestamp-based synchronization.

ULOrderIDPool

The ULOrderIDPool table is a primary key pool for ULOrder.

In the remote database, ULOrderIDPool has the following column:

- **pool_order_id** A primary key column that holds a unique integer identifying the order ID.

In the consolidated database, ULOrderIDPool has the following additional columns:

- **pool_emp_id** An integer column containing the employee ID of the owner of the remote database to which the order ID has been assigned.
- **last_modified** A timestamp containing the last time the row was modified.

ULCustomerIDPool

The ULCustomerIDPool table is a primary key pool for ULCustomer.

In the remote database, ULCustomerIDPool has the following column:

- **pool_cust_id** A primary key column that holds a unique integer identifying the customer ID.

In the consolidated database, ULCustomerIDPool has the following additional columns:

- **pool_emp_id** An integer column containing the employee ID that is used for a new employee generated at a remote database.
- **last_modified** A timestamp containing the last time the row was modified.

The following tables are contained in the consolidated database only:

ULIdentifyEmployee_nosync

The ULIdentifyEmployee_nosync table exists only in the consolidated database. It has a single column as follows:

- **emp_id** This primary key column contains an integer representing an employee ID.

ULEmployee

The ULEmployee table exists only in the consolidated database. It contains a list of sales employees.

ULEmployee has the following columns:

- **emp_id** A primary key column that holds a unique integer identifying the employee.
- **emp_name** A 30-character string containing the name of the employee.

ULEmpCust

The ULEmpCust table controls which customers' orders are downloaded. If the employee needs a new customer's orders, inserting the employee ID and customer ID forces the orders for that customer to be downloaded.

- **emp_id** A foreign key to ULEmployee.emp_id.
- **cust_id** A foreign key to ULCustomer.cust_id. The primary key consists of emp_id and cust_id.
- **action** A character used to determine if an employee record should be deleted from the remote database. If the employee no longer requires a customer's orders, set to D (delete). If the orders are still required, the action should be set to null.

  A logical delete must be used in this case so that the consolidated database can identify which rows to remove from the ULOrder table. Once the deletes have been downloaded, all records for that employee with an action of D can also be removed from the consolidated database.
- **last_modified** A timestamp containing the last time the row was modified. This column is used for timestamp-based synchronization.

ULOldOrder and ULNewOrder

These tables exists only in the consolidated database. They are for conflict resolution and contain the same columns as ULOrder. In SQL Anywhere and Microsoft SQL Server, these are temporary tables. In Adaptive Server Enterprise, these are normal tables and @@spid. DB2 LUW and Oracle do not have temporary tables, so MobiLink needs to be able to identify which rows belong to the synchronizing user. Since these are base tables, if five users are synchronizing, they might each have a row in these tables at the same time.

For more information about @@spid, see "Variables" [*SQL Anywhere Server - SQL Reference*].

Users in the CustDB sample

There are two types of users in the CustDB sample, sales people and mobile managers. The differences are as follows:

- **Sales people** User IDs 50, 51, and 52 identify remote databases that are associated with sales people. Sales people can perform the following tasks:
  - View lists of customers and products.
  - Add new customers.
  - Add or delete orders.
  - Scroll through the list of outstanding orders.
  - Synchronize changes with the consolidated database.

- **Mobile managers** User ID 53 identifies the remote database associated with the mobile manager. The mobile manager can perform the same tasks as a sales person. In addition, the mobile manager can perform the following task:
  - Accept or deny orders.

Synchronizing CustDB

The following sections describe the CustDB sample's synchronization logic.

Synchronization logic source code

You can use Sybase Central to inspect the synchronization scripts in the consolidated database.

Script types and events

The *custdb.sql* file adds each synchronization script to the consolidated database by calling ml_add_connection_script or ml_add_table_script.

Example

The following lines in *custdb.sql* add a table-level script for the ULProduct table, which is executed during the download_cursor event. The script consists of a single SELECT statement.

```
call ml_add_table_script(
'CustDB 12.0',
'ULProduct', 'download_cursor',
'SELECT prod_id, price, prod_name FROM ULProduct' )
go
```

Synchronizing orders in the CustDB sample

Business rules

The business rules for the ULOrder table are as follows:

- Orders are downloaded only if they are not approved or the status is null.
- Orders can be modified at both the consolidated and remote databases.
- Each remote database contains only the orders assigned to an employee.

Downloads

Orders can be inserted, deleted, or updated at the consolidated database. The scripts corresponding to these operations are as follows:

- **download_cursor** The first parameter in the download_cursor script is the last download timestamp. It is used to ensure that only rows that have been modified on either the remote or the consolidated database since the last synchronization are downloaded. The second parameter is the employee ID. It is used to determine which rows to download.

The download_cursor script for CustDB is as follows:

```
CALL ULOrderDownload( {ml s.last_table_download}, {ml s.username} )
```

The ULOrderDownload procedure for CustDB is as follows:

```
CREATE PROCEDURE ULOrderDownload ( IN LastDownload timestamp, IN
EmployeeID integer )
BEGIN`
  SELECT o.order_id, o.cust_id, o.prod_id, o.emp_id, o.disc, o.quant,
o.notes, o.status
    FROM ULOrder o, ULEmpCust ec
    WHERE o.cust_id = ec.cust_id
    AND ec.emp_id = EmployeeID
    AND ( o.last_modified >= LastDownload
    OR ec.last_modified >= LastDownload)
    AND ( o.status IS NULL  OR  o.status != 'Approved' )
    AND ( ec.action IS NULL )
END
```

- **download_delete_cursor** The download_delete_cursor script for CustDB is as follows:

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

Uploads

Orders can be inserted, deleted or updated at the remote database. The scripts corresponding to these operations are as follows:

- **upload_insert** The upload_insert script for CustDB is as follows:

```
INSERT INTO ULOrder ( order_id, cust_id, prod_id, emp_id, disc, quant,
notes, status )
   VALUES( {ml r.order_id, r.cust_id, r.prod_id, r.emp_id, r.disc, r.quant,
r.notes, r.status } )
```

- **upload_update** The upload_update script for CustDB is as follows:

```
UPDATE ULOrder
  SET cust_id = {ml r.cust_id},
      prod_id = {ml r.prod_id},
      emp_id = {ml r.emp_id},
      disc = {ml r.disc},
      quant = {ml r.quant},
      notes = {ml r.notes},
      status = {ml r.status}
    WHERE order_id = {ml r.order_id}
```

- **upload_delete** The upload_delete script for CustDB is as follows:

```
DELETE FROM ULOrder WHERE order_id = {ml r.order_id}
```

- **upload_fetch** The upload_fetch script for CustDB is as follows:


```
SELECT order_id, cust_id, prod_id, emp_id, disc, quant, notes, status
   FROM ULOrder WHERE order_id = {ml r.order_id}
```

- **upload_old_row_insert** The upload_old_row_insert script for CustDB is as follows:

```
INSERT INTO ULOldOrder ( order_id, cust_id, prod_id, emp_id, disc, quant,
notes, status )
   VALUES( {ml r.order_id, r.cust_id, r.prod_id, r.emp_id, r.disc, r.quant,
r.notes, r.status } )
```

- **upload_new_row_insert** The upload_new_row_insert script for CustDB is as follows:

```
INSERT INTO ULNewOrder ( order_id, cust_id, prod_id, emp_id, disc, quant,
notes, status )
   VALUES( {ml r.order_id, r.cust_id, r.prod_id, r.emp_id, r.disc, r.quant,
r.notes, r.status } )
```

Conflict resolution

- **resolve_conflict** The resolve_conflict script for CustDB is as follows:

```
CALL ULResolveOrderConflict
```

The ULResolveOrderConflict procedure for CustDB is as follows:

```
CREATE PROCEDURE ULResolveOrderConflict()
BEGIN
  -- approval overrides denial
  IF 'Approved' = (SELECT status FROM ULNewOrder) THEN
    UPDATE ULOrder o
    SET o.status = n.status, o.notes = n.notes
    FROM ULNewOrder n
    WHERE o.order_id = n.order_id;
  END IF;
  DELETE FROM ULOldOrder;
  DELETE FROM ULNewOrder;
END
```

Synchronizing customers in the CustDB sample

Business rules

The business rules governing customers are as follows:

- Customer information can be modified at both the consolidated and remote databases.
- Both the remote and consolidated databases contain a complete listing of customers.

Downloads

Customer information can be inserted or updated at the consolidated database. The script corresponding to these operations is as follows:

- **download_cursor** The following download_cursor script downloads all customers for whom information has changed since the last time the user downloaded information.

```
SELECT cust_id, cust_name FROM ULCustomer WHERE last_modified >= {ml
s.last_table_download}
```

Uploads

Customer information can be inserted, updated, or deleted at the remote database. The scripts corresponding to these operations are as follows:

- **upload_insert** The upload_insert script for CustDB is as follows:

  ```
  INSERT INTO ULCustomer( cust_id, cust_name )
     VALUES( {ml r.cust_id, r.cust_name } )
  ```

- **upload_update** The upload_update script for CustDB is as follows:

  ```
  UPDATE ULCustomer SET cust_name = {ml r.cust_name}
     WHERE cust_id = {ml r.cust_id}
  ```

  Conflict detection is not performed on this table.

- **upload_delete** The upload_delete script for CustDB is as follows:

  ```
  DELETE FROM ULCustomer WHERE cust_id = {ml r.cust_id}
  ```

Synchronizing products in the CustDB sample

Business rules

All rows are downloaded for ULProduct—this is called snapshot synchronization.

See "Snapshot synchronization" [*MobiLink - Server Administration*].

The business rules for the ULProduct table are as follows:

- Products can only be modified at the consolidated database.
- Each remote database contains all the products.

Downloads

Product information can be inserted, deleted, or updated at the consolidated database. The script corresponding to these operations is as follows:

- **download_cursor** The following download_cursor script downloads all the rows and columns of the ULProduct table at each synchronization:

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

Maintaining the customer and order primary key pools

The CustDB sample database uses primary key pools to maintain unique primary keys in the ULCustomer and ULOrder tables. The primary key pools are the ULCustomerIDPool and ULOrderIDPool tables.

ULCustomerIDPool

The following scripts are defined in the ULCustomerIDPool table:

Downloads

- **download_cursor**

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

Uploads

- **upload_insert** The upload_insert script for CustDB is as follows:

```
INSERT INTO ULCustomerIDPool ( pool_cust_id )
   VALUES( {ml r.pool_cust_id} )
```

- **upload_delete** The upload_delete script for CustDB is as follows:

```
DELETE FROM ULCustomerIDPool
   WHERE pool_cust_id = {ml r.pool_cust_id}
```

- **end_upload** The following end_upload script ensures that after each upload 20 customer IDs remain in the customer ID pool:

```
CALL ULOrderIDPool_maintain( {ml s.username} )
```

  The ULCustomerIDPool_maintain procedure for CustDB is as follows:

```
CREATE PROCEDURE ULCustomerIDPool_maintain ( IN syncuser_id INTEGER )
BEGIN
  DECLARE pool_count INTEGER;
  -- Determine how many ids to add to the pool
  SELECT COUNT(*) INTO pool_count
    FROM ULCustomerIDPool
    WHERE pool_emp_id = syncuser_id;
  -- Top up the pool with new ids
  WHILE pool_count < 20 LOOP
    INSERT INTO ULCustomerIDPool ( pool_emp_id )
      VALUES ( syncuser_id );
    SET pool_count = pool_count + 1;
  END LOOP;
END
```

ULOrderIDPool

The following scripts are defined in the ULOrderIDPool table:

Downloads

- **download_cursor** The download_cursor script for CustDB is as follows:

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

Uploads

- **end_upload** The following end_upload script ensures that after each upload 20 order IDs remain in the order ID pool.

  ```
  CALL ULOrderIDPool_maintain( {ml s.username} )
  ```

  The ULOrderIDPool_maintain procedure for CustDB is as follows:

  ```
  ALTER PROCEDURE ULOrderIDPool_maintain ( IN syncuser_id INTEGER )
  BEGIN
    DECLARE pool_count INTEGER;
    -- Determine how many ids to add to the pool
    SELECT COUNT(*) INTO pool_count
      FROM ULOrderIDPool
      WHERE pool_emp_id = syncuser_id;
    -- Top up the pool with new ids
    WHILE pool_count < 20 LOOP
      INSERT INTO ULOrderIDPool ( pool_emp_id )
        VALUES ( syncuser_id );
      SET pool_count = pool_count + 1;
    END LOOP;
  END
  ```

- **upload_insert** The upload_insert script for CustDB is as follows:

  ```
  INSERT INTO ULOrderIDPool ( pool_order_id )
     VALUES( {ml r.pool_order_id}
  ```

- **upload_delete** The upload_delete script for CustDB is as follows:

  ```
  DELETE FROM ULOrderIDPool
     WHERE pool_order_id = {ml r.pool_order_id}
  ```

Cleanup

To restart the sample, run the following command from the *samples-dir\UltraLite\CustDB* directory:

```
makedbs
```

Further reading

For more information about script types, see "Script types" [*MobiLink - Server Administration*].

For reference material, including details about each script and its parameters, see "Synchronization events" [*MobiLink - Server Administration*].

Exploring the MobiLink Contact sample

Introduction to the Contact sample tutorial

The Contact sample is a valuable resource for the MobiLink developer. It provides you with an example of how to implement many of the techniques you need to develop MobiLink applications.

The Contact sample application includes a SQL Anywhere consolidated database and two SQL Anywhere remote databases. It illustrates several common synchronization techniques. To get the most out of this section, study the sample application as you read.

Although the consolidated database is a SQL Anywhere database, the synchronization scripts consist of SQL statements that should work with minimal changes on other database management systems.

The Contact sample is in *samples-dir\MobiLink\Contact*. For an overview, see the readme in the same location. (For information about *samples-dir*, see "Samples directory" [*SQL Anywhere Server - Database Administration*].)

Synchronization design

The synchronization design in the Contact sample application uses the following features:

- **Column subsets** A subset of the columns of the Customer, Product, SalesRep, and Contact tables on the consolidated database are shared with the remote databases.
- **Row subsets** All the columns, but only one of the rows of the SalesRep table on the consolidated database are shared with each remote database. See "Partitioning rows among remote databases" [*MobiLink - Server Administration*].
- **Timestamp-based synchronization** This is a way of identifying changes that were made to the consolidated database since the last time a device synchronized. The Customer, Contact, and Product tables are synchronized using a method based on timestamps. See "Timestamp-based downloads" [*MobiLink - Server Administration*].

Contact sample setup

A Windows batch file called *build.bat* is provided to build the Contact sample databases. On Unix systems, the file is *build.sh*. You may want to examine the contents of the batch file. It performs the following actions:

- Creates ODBC data source definitions for a consolidated database and each of two remote databases.
- Creates a consolidated database named *consol.db* and loads the MobiLink system tables, database schema, some data, synchronization scripts, and MobiLink user names into the database.
- Creates two remote databases, each named *remote.db*, in subdirectories named *remote_1* and *remote_2*. Loads information common to both databases and applies customizations. These customizations include a global database identifier, a MobiLink user name, and subscriptions to two publications.

To build the Contact sample

1. At a command prompt, navigate to *samples-dir\MobiLink\Contact*.
2. Run *build.bat* (Windows) or *build.sh* (Unix).

For information about *samples-dir*, see "Samples directory" [*SQL Anywhere Server - Database Administration*].

Running the Contact sample

The Contact sample includes batch files that perform initial synchronizations and illustrate MobiLink server and dbmlsync command lines. You can examine the contents of the following batch files, located in *samples-dir\MobiLink\Contact*, in a text editor:

- *step1.bat*
- *step2.bat*
- *step3.bat*

To run the Contact sample

1. Start the MobiLink server.
   a. At a command prompt, navigate to *samples-dir\MobiLink\Contact*.
   b. Run the following command:

      ```
      step1
      ```

      This command runs a batch file that starts the MobiLink server in a verbose mode. This mode is useful during development or troubleshooting, but has a significant performance impact and so would not be used in a routine production environment.
2. Synchronize both remote databases.
   a. At a command prompt, navigate to *samples-dir\MobiLink\Contact*.
   b. Run the following command:

      ```
      step2
      ```

      This is a batch file that synchronizes both remote databases.
3. Shut down the MobiLink server.
   a. At a command prompt, navigate to *samples-dir\MobiLink\Contact*.
   b. Run the following command:

      ```
      step3
      ```

      This is a batch file that shuts down the MobiLink server.

To explore how synchronization works in the Contact sample, you can use Interactive SQL to modify the data in the remote and consolidated databases, and use the batch files to synchronize.

Tables in the Contact databases

The table definitions for the Contact database are located in the following files, all under your samples directory:

- *MobiLink\Contact\build_consol.sql*
- *MobiLink\Contact\build_remote.sql*

Both the consolidated and the remote databases contain the following three tables, although their definition is slightly different in each place.

SalesRep

Each sales representative occupies one row in the SalesRep table. Each remote database belongs to a single sales representative.

In each remote database, SalesRep has the following columns:

- **rep_id** A primary key column that contains an identifying number for the sales representative.
- **name** The name of the representative.

In the consolidated database only, there is also an ml_username column holding the MobiLink user name for the representative.

Customer

This table holds one row for each customer. Each customer is a company with which a single sales representative does business. There is a one-to-many relationship between the SalesRep and Customer tables.

In each remote database, Customer has the following columns:

- **cust_id** A primary key column holding an identifying number for the customer.
- **name** The customer name. This is a company name.
- **rep_id** A foreign key column that references the SalesRep table. Identifies the sales representative assigned to the customer.

In the consolidated database, there are two additional columns, last_modified and active:

- **last_modified** The last time the row was modified. This column is used for timestamp-based synchronization.

- **active** A BIT column that indicates if the customer is currently active (1) or if the company no longer deals with this customer (0). If the column is marked inactive (0) all rows corresponding to this customer are deleted from remote databases.

Contact

This table holds one row for each contact. A contact is a person who works at a customer company. There is a one-to-many relationship between the Customer and Contact tables.

In each remote database, Contact has the following columns:

- **contact_id** A primary key column holding an identifying number for the contact.
- **name** The name of the individual contact.
- **cust_id** The identifier of the customer for whom the contact works.

In the consolidated database, the table also has the following columns:

- **last_modified** The last time the row was modified. This column is used for timestamp-based synchronization.
- **active** A BIT column that indicates if the contact is currently active (1) or if the company no longer deals with this contact (0). If the column is marked inactive (0) the row corresponding to this contact is deleted from remote databases.

Product

Each product sold by the company occupies one row in the Product table. The Product table is held in a separate publication so that remote databases can synchronize the table separately.

In each remote database, Product has the following columns:

- **id** A primary key column that contains a number to identify the product.
- **name** The name of the item.
- **size** The size of the item.
- **quantity** The number of items in stock. When a sales representative takes an order, this column is updated.
- **unit_price** The price per unit of the product.

In the consolidated database, the Product table has the following additional columns:

- **supplier** The company that manufactures the product.
- **last_modified** The last time the row was modified. This column is used for timestamp-based synchronization.

- **active** A BIT column that indicates if the product is currently active (1). If the column is marked inactive (0), the row corresponding to this product is deleted from remote databases.

In addition to these tables, a set of tables is created at the consolidated database only. These include the product_conflict table, which is a temporary table used during conflict resolution, and a set of tables for monitoring MobiLink activities owned by a user named mlmaint. Scripts to create the MobiLink monitoring tables are in the file *samples-dir\MobiLink\Contact\mlmaint.sql*.

Users in the Contact sample

The Contact sample includes several different database user IDs and MobiLink user names.

Database user IDs

The two remote databases are assigned to the sales representatives Samuel Singer (rep_id 856) and Pamela Savarino (rep_id 949).

When connecting to their remote database, both users use the default SQL Anywhere user ID **dba** and the password **SQL**.

Each remote database also has a user ID **sync_user** with the password **sync_user**. This user ID is employed only on the dbmlsync command line. The **sync_user** has REMOTE DBA authority, and can perform any operation when connected from dbmlsync, but has no authority when connected from any other application. So, using the **sync_user** ID and password should not be a problem.

At the consolidated database, there is a user named **mlmaint**, who owns the tables for monitoring MobiLink synchronization statistics and errors. The **mlmaint** user has no right to connect. The assignment of the tables to a separate user ID is done simply to separate the objects from the others in the schema for easier administration in Sybase Central and other utilities.

MobiLink user names

MobiLink user names are distinct from database user IDs. Each remote device has a MobiLink user name in addition to the user ID they use when connecting to a database. The MobiLink user name for Samuel Singer is SSinger. The MobiLink user name for Pamela Savarino is PSavarino. The MobiLink user name is stored or used in the following locations:

- At the remote database, the MobiLink user name is added using a CREATE SYNCHRONIZATION USER statement.
- At the consolidated database, the MobiLink user name and password are added using the mluser utility.
- During synchronization, the MobiLink password for the connecting user is supplied on the dbmlsync command line listed in *MobiLink\Contact\step2.bat*.
- The MobiLink server supplies the MobiLink user name as a parameter to many of the scripts during synchronization.

- The SalesRep table at the consolidated database has an ml_username column. The synchronization scripts match the MobiLink user name parameter against the value in this column.

Synchronizing the Contact sample

The following sections describe the Contact sample's synchronization logic.

Synchronizing sales representatives in the Contact sample

The synchronization scripts for the SalesRep table illustrates **snapshot synchronization**. Regardless of whether a sales representative's information has changed, it is downloaded.

See "Snapshot synchronization" [*MobiLink - Server Administration*].

Business rules

The business rules for the SalesRep table are as follows:

- The table must not be modified at the remote database.
- A sales representative's MobiLink user name and rep_id value must not change.
- Each remote database contains a single row from the SalesRep table, corresponding to the remote database owner's MobiLink user name.

Downloads

- **download_cursor** At each remote database, the SalesRep table contains a single row. There is very little overhead for the download of a single row, so a simple snapshot download_cursor script is used:

  ```
  SELECT rep_id, name
  FROM SalesRep
  WHERE ? IS NOT NULL
  AND ml_username = ?
  ```

  The first parameter in the script is the last download timestamp, which is not used. The IS NOT NULL expression is a dummy expression supplied to use the parameter. The second parameter is the MobiLink user name.

Uploads

This table should not be updated at the remote database, so there are no upload scripts for the table.

Synchronizing customers in the Contact sample

The synchronization scripts for the Customer table illustrate **timestamp-based synchronization** and partitioning rows. Both of these techniques minimize the amount of data that is transferred during synchronization while maintaining consistent table data.

See:

- "Timestamp-based downloads" [*MobiLink - Server Administration*]
- "Partitioning rows among remote databases" [*MobiLink - Server Administration*]

Business rules

The business rules governing customers are as follows:

- Customer information can be modified at both the consolidated and remote databases.
- Periodically, customers may be reassigned among sales representatives. This process is commonly called territory realignment.
- Each remote database contains only the customers they are assigned to.

Downloads

- **download_cursor** The following download_cursor script downloads only active customers for whom information has changed since the last successful download. It also filters customers by sales representative.

  ```
  SELECT cust_id, Customer.name, Customer.rep_id
  FROM Customer key join SalesRep
  WHERE Customer.last_modified >= ?
  AND SalesRep.ml_username = ?
  AND Customer.active = 1
  ```

- **download_delete_cursor** The following download_delete_cursor script downloads only customers for whom information has changed since the last successful download. It deletes all customers marked as inactive or who are not assigned to the sales representative.

  ```
  SELECT cust_id
  FROM Customer key join SalesRep
  WHERE Customer.last_modified >= ?
  AND ( SalesRep.ml_username != ? OR Customer.active = 0 )
  ```

  If rows are deleted from the Customer table at the consolidated database, they do not appear in this result set and so are not deleted from remote databases. Instead, customers are marked as inactive.

  When territories are realigned, this script deletes those customers no longer assigned to the sales representative. It also deletes customers who are transferred to other sales representatives. Such additional deletes are flagged with a SQLCODE of 100 but do not interfere with synchronization. A more complex script could be developed to identify only those customers transferred away from the current sales representative.

  The MobiLink client performs cascading deletes at the remote database, so this script also deletes all contacts who work for customers assigned to some other sales representative.

Uploads

Customer information can be inserted, updated, or deleted at the remote database. The scripts corresponding to these operations are as follows:

- **upload_insert** The following upload_insert script adds a row to the Customer table, marking the customer as active:

```
INSERT INTO Customer(
 cust_id, name, rep_id, active )
VALUES ( ?, ?, ?, 1 )
```

- **upload_update** The following upload_update script modifies the customer information at the consolidated database. Conflict detection is not done on this table.

```
UPDATE Customer
SET name = ?, rep_id = ?
WHERE cust_id = ?
```

- **upload_delete** The following upload_delete script marks the customer as inactive at the consolidated database. It does not delete a row.

```
UPDATE Customer
SET active = 0
WHERE cust_id = ?
```

Synchronizing contacts in the Contact sample

The Contact table contains the name of a person working at a customer company, a foreign key to the customer, and a unique integer identifying the contact. It also contains a last_modified timestamp and a marker to indicate whether the contact is active.

Business rules

The business rules for this table are as follows:

- Contact information can be modified at both the consolidated and remote databases.
- Each remote database contains only those contacts who work for customers they are assigned to.
- When customers are reassigned among sales representatives, contacts must also be reassigned.

Trigger

A trigger on the Customer table is used to ensure that the contacts get picked up when information about a customer is changed. The trigger explicitly alters the last_modified column of each contact whenever the corresponding customer is altered:

```
CREATE TRIGGER UpdateCustomerForContact
AFTER UPDATE OF rep_id ORDER 1
ON DBA.Customer
REFERENCING OLD AS old_cust NEW as new_cust
FOR EACH ROW
BEGIN
  UPDATE Contact
  SET Contact.last_modified = new_cust.last_modified
  FROM Contact
  WHERE Contact.cust_id = new_cust.cust_id
END
```


By updating all contact records whenever a customer is modified, the trigger ties the customer and their associated contacts together. Whenever a customer is modified, all associated contacts are modified too, and the customer and associated contacts are downloaded together on the next synchronization.

Downloads

- **download_cursor** The download_cursor script for Contact is as follows:

```
SELECT contact_id, contact.name, contact.cust_id
FROM ( contact JOIN customer ) JOIN salesrep
ON contact.cust_id = customer.cust_id
 AND customer.rep_id = salesrep.rep_id
WHERE Contact.last_modified >= ?
 AND salesrep.ml_username = ?
 AND Contact.active = 1
```

  This script retrieves all contacts that are active, that have been changed since the last time the sales representative downloaded (either explicitly or by modification of the corresponding customer), and that are assigned to the representative. A join with the Customer and SalesRep table is needed to identify the contacts associated with this representative.

- **download_delete_cursor** The download_delete_cursor script for Contact is as follows:

```
SELECT contact_id
FROM ( Contact JOIN Customer ) JOIN SalesRep
ON Contact.cust_id = Customer.cust_id
 AND Customer.rep_id = SalesRep.rep_id
WHERE Contact.last_modified >= ?
 AND Contact.active = 0
```

  The automatic use of cascading referential integrity by the MobiLink client deletes contacts when the corresponding customer is deleted from the remote database. The download_delete_cursor script therefore has to delete only those contacts marked as inactive.

Uploads

Contact information can be inserted, updated, or deleted at the remote database. The scripts corresponding to these operations are as follows:

- **upload_insert** The following upload_insert script adds a row to the Contact table, marking the contact as active:

```
INSERT INTO Contact (
 contact_id, name, cust_id, active )
VALUES ( ?, ?, ?, 1 )
```

- **upload_update** The following upload_update script modifies the contact information at the consolidated database:

```
UPDATE Contact
SET name = ?, cust_id = ?
WHERE contact_id = ?
```

  Conflict detection is not done on this table.

- **upload_delete** The following upload_delete script marks the contact as inactive at the consolidated database. It does not delete a row.

```
UPDATE Contact
SET active = 0
WHERE contact_id = ?
```

Synchronizing products in the Contact sample

The scripts for the Product table illustrate conflict detection and resolution.

The Product table is kept in a separate publication from the other tables so that it can be downloaded separately. For example, if the price changes and the sales representative is synchronizing over a slow link, they can download the product changes without uploading their own customer and contact changes.

Business rules

The only change that can be made at the remote database is to change the quantity column, when an order is taken.

Downloads

- **download_cursor** The following download_cursor script downloads all rows changed since the last time the remote database synchronized:

  ```
  SELECT id, name, size, quantity, unit_price
  FROM product
  WHERE last_modified >= ?
  AND active = 1
  ```

- **download_delete_cursor** The following download_delete_cursor script removes all products no longer sold by the company. These products are marked as inactive in the consolidated database.

  ```
  SELECT id, name, size, quantity, unit_price
  FROM product
  WHERE last_modified >= ?
  AND active = 0
  ```

Uploads

Only UPDATE operations are uploaded from the remote database. The major feature of these upload scripts is a conflict detection and resolution procedure.

If two sales representatives take orders and then synchronize, each order is subtracted from the quantity column of the Product table. For example, if Samuel Singer takes an order for 20 baseball hats (product ID 400), he changes the quantity from 90 to 70. If Pamela Savarino takes an order for 10 baseball hats before receiving this change, she changes the column in her database from 90 to 80.

When Samuel Singer synchronizes his changes, the quantity column in the consolidated database is changed from 90 to 70. When Pamela Savarino synchronizes her changes, the correct action is to set the value to 60. This setting is accomplished by detecting the conflict.

The conflict detection scheme includes the following scripts:

- **upload_update** The following upload_update script is a straightforward UPDATE at the consolidated database:


```
UPDATE product
SET name = ?, size = ?, quantity = ?, unit_price = ?
WHERE product.id = ?
```

- **upload_fetch** The following upload_fetch script fetches a single row from the Product table for comparison with the old values of the uploaded row. If the two rows differ, a conflict is detected.

```
SELECT id, name, size, quantity, unit_price
FROM Product
WHERE id = ?
```

- **upload_old_row_insert** If a conflict is detected, the old values are placed into the product_conflict table for use by the resolve_conflict script. The row is added with a value of O (for Old) in the row_type column.

```
INSERT INTO DBA.product_conflict(
 id, name, size, quantity, unit_price, row_type )
VALUES( ?, ?, ?, ?, ?, 'O' )' )
```

- **upload_new_row_insert** The following script adds the new values of the uploaded row into the product_conflict table for use by the resolve_conflict script:

```
INSERT INTO DBA.product_conflict(
 id, name, size, quantity, unit_price, row_type )
VALUES( ?, ?, ?, ?, ?, 'N' )
```

Conflict resolution

- **resolve_conflict** The following script resolves the conflict by adding the difference between new and old rows to the quantity value in the consolidated database:

```
UPDATE Product
SET p.quantity = p.quantity
              - old_row.quantity
              + new_row.quantity
FROM Product p,
     DBA.product_conflict old_row,
     DBA.product_conflict new_row
WHERE p.id = old_row.id
    AND p.id = new_row.id
    AND old_row.row_type = 'O'
    AND new_row.row_type = 'N'
```

Monitoring statistics and errors in the Contact sample

The Contact sample contains some simple error reporting and monitoring scripts. The SQL statements to create these scripts are in the file *MobiLink\Contact\mlmaint.sql*.

The scripts insert rows into tables created to hold the values. For convenience, the tables are owned by a distinct user, mlmaint.

MobiLink Tutorials

This section provides tutorials that show you how to set up and use MobiLink technology. These range from very introductory tutorials for new users to demonstrations of how to use advanced features.

Tutorial: Introduction to MobiLink

Introduction to MobiLink synchronization

This tutorial guides you through the basic steps for writing synchronization scripts, error-checking using MobiLink logs, and monitoring synchronizations between a consolidated database and two remote databases using the MobiLink Monitor. It provides instructions for setting up the databases and synchronizations using Sybase Central.

Required software

- SQL Anywhere 12

Competencies and experience

You require:

- Basic knowledge of MobiLink event scripts and MobiLink synchronization

Goals

You gain competence and familiarity with:

- Migrating the consolidated database schema to remote databases.
- Writing the basic scripts needed for synchronization and storing them in the consolidated database using Sybase Central or Interactive SQL.
- Writing scripts for conflict detection and resolution.
- Monitoring synchronization using log files and the MobiLink Monitor.

Suggested background reading

- "Understanding MobiLink synchronization" on page 1
- "Writing synchronization scripts" [*MobiLink - Server Administration*]
- "Using Interactive SQL" [*SQL Anywhere Server - Database Administration*]
- "Using Sybase Central" [*SQL Anywhere Server - Database Administration*]
- "Handling conflicts" [*MobiLink - Server Administration*]
- "MobiLink Monitor" [*MobiLink - Server Administration*]

MobiLink code exchange samples are located at http://www.sybase.com/detail?id=1058600#319.

You can post questions on the MobiLink newsgroup: sybase.public.sqlanywhere.mobilink.

Lesson 1: Set up your MobiLink consolidated database

This lesson guides you through the following steps to set up your SQL Anywhere consolidated database:

1. Create the consolidated database and schema.
2. Run the MobiLink setup script.
3. Define an ODBC data source for the consolidated database.

Create your consolidated database

In this tutorial, you create a SQL Anywhere database using the Sybase Central **Create Database Wizard**.

To create your SQL Anywhere RDBMS

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.
2. In Sybase Central, choose **Tools** » **SQL Anywhere 12** » **Create Database**.
3. Click **Next**.
4. Leave the default of **Create Database On This Computer**, and then click **Next**.
5. In the **Save The Main Database File To The Following File** field, type the file name and path for the database. For example, *c:\MLmon\MLconsolidated.db*. Click **Next**.
6. Follow the remaining instructions in the **Create Database Wizard** and accept the default values. On the **Connect To The Database** page, clear the **Stop The Database After Last Disconnect** option.
7. Click **Finish**.

   The **MLconsolidated** database appears in Sybase Central.
8. Click **Close** on the **Creating Database** window.

Define an ODBC data source for the consolidated database

Use the SQL Anywhere 12 driver to define an ODBC data source for the **MLconsolidated** database.

To define an ODBC data source for the consolidated database

1. From the Sybase Central **Tools** menu, choose **SQL Anywhere 12** » **Open ODBC Administrator**.
2. Click the **User DSN** tab, and click **Add**.
3. In the **Create New Data Source** window, click **SQL Anywhere 12** and click **Finish**.
4. Perform the following tasks in the **ODBC Configuration For SQL Anywhere** window:
   a. Click the **ODBC** tab.
   b. In the **Data Source Name** field, type **mlmon_db**.

c. Click the **Login** tab.

d. In the **Authentication** dropdown list, choose **Database** to connect using your user ID and password.

e. In the **User ID** field, type **DBA**.

f. In the **Password** field, type **sql**.

g. From the **Action** dropdown list, choose **Connect To A Running Database On This Computer**.

h. In the **Server Name** field, type **MLconsolidated**.

i. Click **OK**.

5. Close the ODBC data source administrator.

   Click **OK** on the **ODBC Data Source Administrator** window.

Set up tables for synchronization

In this procedure, you create the Product table in the MobiLink consolidated database. The Product table contains the following columns:

Column	Description
name	The name of the product.
quantity	The number of items sold.
last_modified	The last modification date of a row. You use this column for timestamp based downloads, a common technique used to filter rows for efficient synchronization.

You create temporary tables for conflict resolution purposes in addition to the Product table. These tables are named Product_old, and Product_new.

To create the tables

1. Connect to your database using Interactive SQL.

   You can start Interactive SQL from Sybase Central or from a command prompt.

   - To start Interactive SQL from Sybase Central, right-click the **MLconsolidated - DBA** database and click **Open Interactive SQL**.
   - To start Interactive SQL at a command prompt, run the following command:

     ```
     dbisql -c "dsn=mlmon_db"
     ```

2. Run the following SQL script in Interactive SQL to create the **Product** table:

   ```
   CREATE TABLE Product (
       name            VARCHAR(128) NOT NULL PRIMARY KEY,
       quantity        INTEGER,
       last_modified   TIMESTAMP DEFAULT TIMESTAMP
   )
   ```

Interactive SQL creates the **Product** table in your consolidated database.

3. Run the following SQL script in Interactive SQL to create the temporary tables:

```
CREATE TABLE Product_old (
    name            VARCHAR(128) NOT NULL PRIMARY KEY,
    quantity        INTEGER,
    last_modified   TIMESTAMP DEFAULT TIMESTAMP
)
CREATE TABLE Product_new (
    name            VARCHAR(128) NOT NULL PRIMARY KEY,
    quantity        INTEGER,
    last_modified   TIMESTAMP DEFAULT TIMESTAMP
)
```

Interactive SQL creates the **Product_old** and **Product_new** tables in your consolidated database.

After creating the tables, you must populate the **Product** table.

To populate the Product table

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   You can start Interactive SQL from Sybase Central or from a command prompt.

   - To start Interactive SQL from Sybase Central, right-click the **MLconsolidated - DBA** database and click **Open Interactive SQL**.
   - To start Interactive SQL at a command prompt, run the following command:

     ```
     dbisql -c "dsn=mlmon_db"
     ```

2. Run the following SQL script in Interactive SQL to create the **Product** table:

```
INSERT INTO Product(name, quantity)
    VALUES ( 'Screwmaster Drill', 10);

INSERT INTO Product(name, quantity)
    VALUES ( 'Drywall Screws 10lb', 30);

INSERT INTO Product(name, quantity)
    VALUES ( 'Putty Knife x25', 12);

COMMIT;
```

3. Verify that the **Product** table contains product information.

   Run the following SQL script to verify the contents:

   ```
   SELECT * FROM Product
   ```

   The contents of the **Product** table should appear in Interactive SQL.

Stay connected in Interactive SQL for the next procedure.

Run MobiLink setup scripts

You can find setup scripts for each supported consolidated database in the *MobiLink/setup* subdirectory of your SQL Anywhere 12 installation.

In this procedure you set up a SQL Anywhere consolidated database. You do this using the *syncsa.sql* setup script. Running *syncsa.sql* creates a series of system tables and stored procedures prefaced with **ml_**. The MobiLink server works with these tables and stored procedures in the synchronization process.

To install MobiLink system tables

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "dsn=mlmon_db"
   ```

2. Run the following command in Interactive SQL to create MobiLink system tables and stored procedures. Replace *C:\Program Files\SQL Anywhere 12* with the location of your SQL Anywhere 12 installation.

   ```
   READ "C:\Program Files\SQL Anywhere 12\MobiLink\setup\syncsa.sql"
   ```

   Interactive SQL applies *syncsa.sql* to your consolidated database.

Stay connected in Interactive SQL for the next lesson.

Further reading

For information about using the dbinit command line utility to create your consolidated database, see "Initialization utility (dbinit)" [*SQL Anywhere Server - Database Administration*] or "Lesson 2: Add synchronization scripts" on page 77.

For information about creating tables, see "CREATE TABLE statement" [*SQL Anywhere Server - SQL Reference*].

For information about setting up MobiLink consolidated databases, see "MobiLink consolidated databases" [*MobiLink - Server Administration*].

For more information about Interactive SQL, see "Using Interactive SQL" [*SQL Anywhere Server - Database Administration*].

Lesson 2: Add synchronization scripts

You can view, write, and modify synchronization scripts using Sybase Central. In this section you write the following synchronization scripts:

- **upload_insert** This event defines how new client-side data should be applied to the consolidated database.
- **download_cursor** This event defines the data that should be downloaded to remote clients.
- **download_delete_cursor** This event is required when using synchronization scripts that are not upload-only. You set the MobiLink server to ignore this event for the purpose of this tutorial.

Connect to the consolidated database by creating a new MobiLink project.

To create a new MobiLink project

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.
2. From the **Tools** menu, choose **MobiLink 12** » **New Project**.
3. In the **Name** field, type **mlmon_project**.
4. In the **Location** field, type *C:\mlmon*, and then click **Next**.
5. Check the **Add A Consolidated Database To The Project** option.
6. In the **Database Display Name** field, type **mlmon_db**.
7. Click **Edit**. The **Connect To A Generic ODBC Database** window appears.
8. In the **User ID** field, type **DBA**.
9. In the **Password** field, type **sql**.
10. In the **ODBC Data Source name** field, click **Browse** and select **mlmon_db**.
11. Click **OK**, then click **Save**.
12. Check the **Remember The Password** option, and then click **Finish**.
13. Click **OK**.

Each script belongs to a designated script version. You must add a script version to the consolidated database before you add scripts.

To add a script version

1. From the **View** menu, choose **Folders**.
2. In the left pane of Sybase Central under **MobiLink 12**, expand **mlmon_project**, **Consolidated Databases** and then **mlmon_db - DBA**.
3. Right-click **Versions** and choose **New** » **Version**.
4. In the **What Do You Want To Name The New Script Version** field, type **ver1**.
5. Click **Finish**.

You must register the Product table as a synchronization table before you can add scripts to that table.

To add synchronized tables to your consolidated database

1. From the **View** menu, choose **Folders**.

2. In the left pane of Sybase Central under **MobiLink 12**, expand **mlmon_project**, **Consolidated Databases** and then **mlmon_db - DBA**.

3. Right-click **Synchronized Tables** and choose **New** » **Synchronized Table**.

4. Click the **Choose A Table In The Consolidated Database With The Same Name As The Remote Table** option.

5. In the **Which User Owns The Table You Want To Synchronize** list, click **DBA**.

6. In the **Which Table Do You Want To Synchronize** list, click **Product**.

7. Click **Finish**.

Add a new table script for each upload and download to the consolidated database.

To add table scripts for the Product table

1. From the **View** menu, choose **Folders**.

2. In the left pane of Sybase Central under **MobiLink 12**, expand **mlmon_project**, **Consolidated Databases**, **mlmon_db** and then **Synchronized Tables**.

3. Right-click **Product** and choose **New** » **Table Script**.

4. In the **For Which Version Do You Want To Create The Table Script** list, click **ver1**.

5. In the **Which Event Should Cause The Table Script To Be Executed** list, click **upload_insert**, and then click **Next**.

6. Click **Finish**.

7. In the right pane of Sybase Central, use the following SQL script for the **upload_insert** event:

   ```
   INSERT INTO Product( name, quantity, last_modified )
       VALUES( {ml r.name}, {ml r.quantity}, now() )
   ```

   The upload_insert event determines how data inserted into the remote database should be applied to the consolidated database. For more information about upload_insert, see "upload_insert table event" [*MobiLink - Server Administration*].

8. Choose **File** » **Save**.

9. Repeat steps 2 to 5, specifying the **download_cursor** event instead of the **upload_insert** event in step 4.

10. In the right pane of Sybase Central, use the following SQL script for the **download_cursor** event:

    ```
    SELECT name, quantity FROM Product
        WHERE last_modified >= {ml s.last_table_download}
    ```

The download_cursor script defines a cursor to select consolidated database rows that are downloaded and inserted or updated in the remote database. For more information about download_cursor, see "download_cursor table event" [*MobiLink - Server Administration*].

11. Choose **File** » **Save**.

12. Repeat steps 2 to 5, specifying the **download_delete_cursor** event instead of the **upload_insert** event in step 4.

13. In the right pane of Sybase Central, use the following SQL script for the **download_delete_cursor** event:

```
--{ml_ignore}
```

14. Choose **File** » **Save**.

Further reading

For information about using SQL-based events to upload data from remote clients to a MobiLink consolidated database, see:

- "Writing scripts to upload rows" [*MobiLink - Server Administration*]
- "upload_insert table event" [*MobiLink - Server Administration*]
- "upload_update table event" [*MobiLink - Server Administration*]
- "upload_delete table event" [*MobiLink - Server Administration*]

For information about uploading data to data sources other than consolidated databases, see "Handling direct uploads" [*MobiLink - Server Administration*].

For information about using SQL-based events to download data from a MobiLink consolidated database, see:

- "Writing scripts to download rows" [*MobiLink - Server Administration*]
- "download_cursor table event" [*MobiLink - Server Administration*]
- "download_delete_cursor table event" [*MobiLink - Server Administration*]

For information about downloading data to data sources other than consolidated databases, see "Handling direct downloads" [*MobiLink - Server Administration*].

For information about the synchronization event sequence, see "Overview of MobiLink events" [*MobiLink - Server Administration*].

For information about synchronization techniques for download filtering, see "Timestamp-based downloads" [*MobiLink - Server Administration*] and "Partitioning rows among remote databases" [*MobiLink - Server Administration*].

For information about managing scripts, see "Adding and deleting scripts" [*MobiLink - Server Administration*].

For information about direct row handling, see "Direct row handling" [*MobiLink - Server Administration*].

Lesson 3: Start the MobiLink server

In this lesson, you start the MobiLink server. You start the MobiLink server (mlsrv12) using the -c option to connect to your consolidated database.

To start the MobiLink server

- Connect to your consolidated database.

  Run the following command:

  ```
  mlsrv12 -c "dsn=mlmon_db" -o serverOut.txt -v+ -dl -zf -zu+ -x tcpip
  ```

  The MobiLink server messages window appears.

Note
The -zf option should be used for debugging and development purposes only. This tutorial requires the -zf option so that you do not need to shut down the server when adding new scripts to the consolidated database in a later lesson.

Below is a description of each MobiLink server option used in this tutorial. The options -o, -v, and -dl provide debugging and troubleshooting information. Using these logging options is appropriate in a development environment. For performance reasons, -v and -dl are typically not used in production.

Option	Description
-c	Precedes the connection string.
-o	Specifies the message log file *serverOut.txt*.
-v+	Specifies what information is logged. Using -v+ sets maximum verbose logging.
-zf	Causes the MobiLink server to check for script changes at the beginning of each synchronization.
-dl	Displays all log messages on screen.
-zu+	Adds new users automatically.
-x	Sets the communications protocol and parameters for MobiLink clients.

Further reading

For a complete list of MobiLink server options, see "MobiLink server options" [*MobiLink - Server Administration*].

Lesson 4: Set up your MobiLink client databases

MobiLink is designed for synchronization involving a consolidated database server and a large number of mobile databases. In this section, you create two remote databases. For each database you must perform the following tasks:

- Create a Product table, which you synchronize with the consolidated database.
- Create a synchronization publication, user, and subscription.

In this lesson, you use a SQL Anywhere database for your consolidated database and your MobiLink client. For tutorial purposes your MobiLink client, consolidated database, and MobiLink server all reside on the same computer.

Set up SQL Anywhere databases

To set up the MobiLink client databases, you create Product tables for each database. The Product table corresponds to the Product table on the consolidated database but does not include the **last_modified** column. The MobiLink server uses SQL-based scripts to synchronize product quantities.

You create a synchronization user, publication, and subscription on the client database after creating the tables. Publications identify the tables and columns on your remote database that you want synchronized. These tables and columns are called **articles**. A synchronization subscription subscribes a MobiLink user to a publication.

To set up your MobiLink client databases

1. Create your MobiLink client databases using the dbinit command line utility.

   Run the following command to create the **remote1** database:

   ```
   dbinit -p 4096 remote1
   ```

   Run the following command to create the **remote2** database:

   ```
   dbinit -p 4096 remote2
   ```

   The -p option defines a 4K page size shown to increase performance for many environments.

2. Start your MobiLink client database using the dbeng12 command line utility.

   Run the following command to start the **remote1** database:

   ```
   dbeng12 remote1
   ```

   Run the following command to start the **remote2** database:

   ```
   dbeng12 remote2
   ```

3. Connect to the **remote1** database using Interactive SQL.

   Run the following command:


```
dbisql -c "server=remote1;uid=DBA;pwd=sql"
```

4. Create the Product table for the **remote1** database.

   Run the following SQL script in Interactive SQL:

   ```
   CREATE TABLE Product (
       name            VARCHAR(128) NOT NULL PRIMARY KEY,
       quantity        INTEGER
   )
   ```

5. Create your MobiLink synchronization user, publication, and subscription for the **remote1** database.

   Run the following SQL script in Interactive SQL:

   ```
   CREATE PUBLICATION pub_1 (TABLE Product);
   CREATE SYNCHRONIZATION USER user_1;
   CREATE SYNCHRONIZATION SUBSCRIPTION TO pub_1 FOR user_1
     TYPE TCPIP ADDRESS 'host=localhost' OPTION scriptversion='ver1';
   ```

6. Close Interactive SQL.

7. Connect to the **remote2** database using Interactive SQL.

   Run the following command:

   ```
   dbisql -c "server=remote2;uid=DBA;pwd=sql"
   ```

8. Create the Product table for the **remote2** database.

   Run the following SQL script in Interactive SQL:

   ```
   CREATE TABLE Product (
       name            VARCHAR(128) NOT NULL PRIMARY KEY,
       quantity        INTEGER
   )
   ```

9. Create your MobiLink synchronization user, publication, and subscription for the **remote2** database.

   Run the following SQL script in Interactive SQL:

   ```
   CREATE PUBLICATION pub_2 (TABLE Product);
   CREATE SYNCHRONIZATION USER user_2;
   CREATE SYNCHRONIZATION SUBSCRIPTION TO pub_2 FOR user_2
     TYPE TCPIP ADDRESS 'host=localhost' OPTION scriptversion='ver1';
   ```

10. Close Interactive SQL.

Further reading

For information about creating a SQL Anywhere database, see "Initialization utility (dbinit)" [*SQL Anywhere Server - Database Administration*].

For information about MobiLink clients, see "MobiLink clients" [*MobiLink - Client Administration*].

For information about creating MobiLink objects on the client, see:

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

Lesson 5: Synchronize

The dbmlsync utility initiates MobiLink synchronization for SQL Anywhere remote databases.

To start the synchronization clients

1. Run the following command to synchronize the **remote1** database:

   ```
   dbmlsync -c "server=remote1;uid=DBA;pwd=sql" -o rem1.txt -v+
   ```

2. Run the following command to synchronize the **remote2** database:

   ```
   dbmlsync -c "server=remote2;uid=DBA;pwd=sql" -o rem2.txt -v+
   ```

The following table contains a description for each dbmlsync option used in this lesson:

Option	Description
-c	Specifies the connection string.
-o	Specifies the message log file.
-v+	The -v option specifies what information is logged. Using -v+ sets maximum verbose logging.

An output screen appears indicating that the synchronization succeeded. SQL-based synchronization transferred rows in the client **Product** table to the **Product** table in the consolidated database.

Further reading

For more information about the dbmlsync, see "SQL Anywhere clients" [*MobiLink - Client Administration*].

Lesson 6: Monitor your MobiLink synchronization using log files

After the tables are synchronized, you can view the progress of the synchronization using the message log files you created with each command line, *serverOut.txt*, *rem1.txt*, and *rem2.txt*. The default location of these files is the directory where the command was run.

To detect errors in your MobiLink synchronization server log file

1. Open your log file in a text editor. For this tutorial, the log file is *serverOut.txt*.
2. Scan down the left side of the file. A line beginning with **I.** contains an informational message, and a line beginning with **E.** contains an error message.

To detect errors in your MobiLink synchronization client log file

1. Open a client log file, such as *rem1.txt* or *rem2.txt*, in a text editor.
2. Search the file for the string `COMMIT`. If it appears, your synchronization was successful.
3. Search the file for the string `ROLLBACK`. If the transaction was rolled back, there were errors that prevented it from completing.
4. Scan down the left side of the file. An error has occurred if you see a line that begins with an **E.**. Your synchronization has completed successfully if your log file does not contain errors.

Example

The following example shows sample error messages that you could encounter when reviewing the MobiLink server log:

```
E. 2010-04-12 14:49:14. Error code from MobiLink server: -10355
E. 2010-04-12 14:49:14. Server error: Message: The table 'Product' has no
upload_update script.  Table Name: Product
```

This message indicates that an error has occurred before data could be uploaded. The message implies that the required upload_update script does not exist in the Product synchronization table of the consolidated database.

Further reading

For more information about MobiLink log files, see "Logging MobiLink server actions" [*MobiLink - Server Administration*].

Lesson 7: Creating scripts for conflict detection and resolution

Conflicts arise during the upload of rows to the consolidated database. If two users modify the same row on different remote databases, a conflict is detected when the second row arrives at the MobiLink server. Using synchronization scripts you can detect and resolve conflicts.

Inventory example

Consider a scenario where two clients are selling up to ten items simultaneously. The first client sells three items, so they update the inventory on their client database, **remote1**, to indicate seven remaining items. Their database then synchronizes with the consolidated database, **mlmon_db**. The second client then sells four items, so they update the inventory on their remote database, **remote2**, to indicate six

remaining items. A conflict is detected when they attempt to synchronize with **mlmon_db** because the inventory value has changed.

You must obtain the following row values to programmatically resolve this conflict:

1. The current value in the consolidated database.

   The value in the consolidated database is 7 after **remote1** synchronizes.

2. The new row value uploaded by the **remote2** database.

3. The old row value obtained by the **remote2** database during the previous synchronization.

You can use the following business logic to calculate the new inventory value and resolve the conflict:

```
current consolidated - (old remote - new remote)
that is, 7 - (10-6) = 3
```

The (old remote - new remote) expression provides the number of items sold by second client rather than the absolute inventory value.

Create synchronization scripts for conflict detection and resolution

You add the following scripts to detect and resolve conflicts:

- **upload_fetch** You use this script to fetch rows from a table in the consolidated database for conflict detection.

- **upload_update** You use this script to determine how data inserted into the remote database should be applied to the consolidated database. You can also use an extended prototype of upload_update to detect update conflicts.

  For more information about upload_update, see "upload_update table event" [*MobiLink - Server Administration*].

- **upload_old_row_insert** You use this script to handle old row values obtained by the remote database during its previous synchronization.

  For more information about upload_old_row_insert, see "upload_old_row_insert table event" [*MobiLink - Server Administration*].

- **upload_new_row_insert** You use this script to handle new row values (the updated values on the remote database).

  For more information about upload_new_row_insert, see "upload_new_row_insert table event" [*MobiLink - Server Administration*].

- **upload_delete** You use this script to handle rows that are deleted from the remote database. You set the MobiLink server to ignore this event for the purpose of this tutorial.

- **resolve_conflict** The resolve conflict script applies business logic to resolve the conflict.

For more information about resolve_conflict, see "resolve_conflict table event" [*MobiLink - Server Administration*].

To install synchronization scripts for conflict detection and resolution

1. Connect to your database using Interactive SQL.

   You can start Interactive SQL from Sybase Central or from a command prompt.

   - To start Interactive SQL from Sybase Central, right-click the **MLconsolidated - DBA** database and click **Open Interactive SQL**.
   - To start Interactive SQL at a command prompt, run the following command:

     ```
     dbisql -c "dsn=mlmon_db"
     ```

2. Install the conflict detection and resolution scripts.

   Run the following SQL script:

   ```
   /* upload_fetch */
   CALL ml_add_table_script( 'ver1', 'Product',
       'upload_fetch',
       'SELECT name, quantity FROM Product WHERE name = {ml r.name}' );

   /* upload_update */
   CALL ml_add_table_script( 'ver1', 'Product',
       'upload_update',
       'UPDATE Product
           SET quantity = {ml r.quantity}, last_modified = now()
           WHERE name = {ml r.name}' );

   /* upload_old_row_insert */
   CALL ml_add_table_script( 'ver1', 'Product',
       'upload_old_row_insert',
       'INSERT INTO Product_old (name,quantity,last_modified)
           VALUES ({ml r.name}, {ml r.quantity}, now())');

   /* upload_new_row_insert */
   CALL ml_add_table_script( 'ver1', 'Product',
       'upload_new_row_insert',
       'INSERT INTO Product_new (name,quantity,last_modified)
           VALUES ({ml r.name}, {ml r.quantity}, now())');

   /* upload_delete */
   CALL ml_add_table_script( 'ver1', 'Product',
       'upload_delete', '--{ml_ignore}');

   /* resolve_conflict */
   CALL ml_add_table_script( 'ver1', 'Product',
       'resolve_conflict',
       'DECLARE @product_name VARCHAR(128);
        DECLARE @old_rem_val INTEGER;
        DECLARE @new_rem_val INTEGER;
        DECLARE @curr_cons_val INTEGER;
        DECLARE @resolved_value INTEGER;

        // obtain the product name
        SELECT name INTO @product_name FROM Product_old;

        // obtain the old remote value
   ```

```
        SELECT quantity INTO @old_rem_val FROM Product_old;

        //obtain the new remote value
        SELECT quantity INTO @new_rem_val FROM Product_new;

        // obtain the current value in cons
        SELECT quantity INTO @curr_cons_val FROM Product WHERE name =
    @product_name;

        // determine the resolved value
        SET @resolved_value = @curr_cons_val- (@old_rem_val - @new_rem_val);

        // update cons with the resolved value
        UPDATE Product
            SET quantity = @resolved_value
                WHERE name = @product_name;

       // clear the old and new row tables
       DELETE FROM Product_new;
       DELETE FROM Product_old');

    COMMIT
```

Note
In this tutorial, the MobiLink server runs with the -zf option specified, which allows the server to detect any new scripts added to the consolidated database during a synchronization. Unless this option is specified, you must stop the MobiLink server before adding new scripts to the consolidated database; restart the server after adding the new scripts.

Further reading

For more information about MobiLink conflict detection and resolution, see "Handling conflicts" [*MobiLink - Server Administration*] and "Detecting conflicts" [*MobiLink - Server Administration*].

For more information about MobiLink consolidated databases, see "MobiLink consolidated databases" [*MobiLink - Server Administration*].

Lesson 8: Use the MobiLink Monitor to detect update conflicts

You can use the MobiLink Monitor to collect statistical information about synchronizations as they occur. The MobiLink Monitor's graphical chart shows tasks on the vertical axis against the progression of time on the horizontal axis.

Using the MobiLink Monitor, you can quickly identify synchronizations that result in errors or satisfy certain conditions. Since the MobiLink Monitor does not significantly degrade performance, it is recommended for both development and production.

In this section you:

- Start and configure the MobiLink Monitor to visibly distinguish synchronizations that involve update conflicts.

- Generate a conflict by updating the same row on the client databases.
- Use the MobiLink Monitor to detect the conflict.

To configure the MobiLink Monitor to detect update conflicts

1. Click **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **MobiLink Monitor**.
2. Connect to the MobiLink server:

   In the **User** field, type **monitor_user**. As you started the MobiLink server with the -zu+ option, this user is added automatically.
3. Click **Tools** » **Watch Manager** to start the MobiLink Monitor Watch Manager.
4. Add a new watch for update conflicts:
   a. Click **New**.
   b. In the **Name** field, type **conflict_detected**.
   c. In the **Property** list, click **conflicted_updates**.

      The conflicted_updates statistical property indicates the number of uploaded updates for which conflicts were detected.

      For more information about MobiLink Monitor statistical properties, see "MobiLink statistical properties" [*MobiLink - Server Administration*].
   d. In the **Operator** list, click **is greater than**.
   e. In the **Value** field, type **0**.
   f. Click **Add**.
   g. In the **Chart Pattern** list, set the pattern for the **Chart** pane. The Chart pane is the middle pane in MobiLink Monitor.
   h. In the **Overview color** list, set the color for the **Overview** pane. The Overview pane is the bottom pane in the MobiLink Monitor.
5. Click **OK**.
6. Click **OK**.

The first clients starts with a Screwmaster Drill inventory of ten items, and then sells three. He updates the inventory on his remote database, **remote1**, to seven items.

To generate an update conflict

1. Connect to the **remote1** database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "server=remote1;uid=DBA;pwd=sql"
   ```

2. Update the Screwmaster Drill quantity to seven.

   Run the following SQL script:

   ```
   UPDATE Product SET quantity = 7
       WHERE name ='Screwmaster Drill'
   COMMIT
   ```

3. Synchronize the **remote1** database with the consolidated database.

   Run the following command:

   ```
   dbmlsync -c "server=remote1;uid=DBA;pwd=sql" -v+
   ```

   The consolidated database updates the Screwmaster Drill quantity to seven.

4. Connect to the **remote2** database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "server=remote2;uid=DBA;pwd=sql"
   ```

5. Update the Screwmaster Drill quantity to six.

   ```
   UPDATE Product SET quantity = 6
       WHERE name ='Screwmaster Drill'
   COMMIT
   ```

6. Synchronize the **remote2** database with the consolidated database.

   Run the following command:

   ```
   dbmlsync -c "server=remote2;uid=DBA;pwd=sql" -v+
   ```

You switch to the MobiLink Monitor and view the results of the synchronization.

To detect the update conflict using the MobiLink Monitor

1. Pause Chart Scrolling.

   Click **Monitor** » **Pause Chart Scrolling**.

2. View statistical information about the synchronization using the MobiLink Monitor **Overview** pane (the bottom pane), **Chart** pane, the **Utilization Graph** pane, and **Details Table** table.

   a. Locate the synchronizations in the MobiLink Monitor **Overview** pane. The **remote2** synchronization which generated an update conflict, appears in red:

   b. To view the **remote2** synchronization in the **Chart** pane, click and drag over the synchronization object in the **Overview** pane.

      The synchronization object is displayed with the pattern you chose for the conflict_detected watch.

   c. Use the zoom tool to view synchronization detail.

      From the **View** menu, choose **Zoom In**.

d. Double-click the synchronization object or the corresponding row in the **Details Table** pane to view synchronization properties.

e. Choose the **Upload** tab to see the number of conflicted updates.

Further reading

For more information about MobiLink conflict resolution, see "Handling conflicts" [*MobiLink - Server Administration*].

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

For more information about MobiLink Monitor statistical properties, see "MobiLink statistical properties" [*MobiLink - Server Administration*].

Cleanup

Remove tutorial materials from your computer.

To remove tutorial materials from your computer

1. Close all instances of the following applications:
   - The MobiLink Monitor
   - Sybase Central
   - Interactive SQL
2. Close the SQL Anywhere, MobiLink, and synchronization client windows by right-clicking each taskbar item and choosing **Close**.
3. Delete all tutorial-related data sources:
   a. Start the ODBC Data Source Administrator.
   b. From the **Start** menu, choose **Programs** » **SQL Anywhere 12** » **Administration Tools** » **ODBC Data Source Administrator**.
   c. Select **mlmon_db** from the list of **User Data Sources**, and click **Remove**.
4. Delete the consolidated and remote databases.
   a. Navigate to the directory containing your consolidated and remote databases.
   b. Delete *MLconsolidated.db*, *MLconsolidated.log*, *remote1.db*, *remote1.log*, *remote2.db*, *remote2.log*.

Further reading

For more information about running the MobiLink server, see "MobiLink server" [*MobiLink - Server Administration*].

For more information about synchronization scripting, see "Writing synchronization scripts" [*MobiLink - Server Administration*] and "Synchronization events" [*MobiLink - Server Administration*].

For an introduction to other methods of synchronization such as timestamp-based synchronization, see "Synchronization techniques" [*MobiLink - Server Administration*].

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

For more information about MobiLink conflict detection and resolution, see "Handling conflicts" [*MobiLink - Server Administration*].

Tutorial: Using MobiLink with an Oracle Database 10g

Introduction to Oracle tutorial

This tutorial shows you to mobilize an Oracle Database 10g using MobiLink. It sets up synchronization between an Oracle Database 10g and a SQL Anywhere remote database. You could also set up an UltraLite remote database.

The purpose of this tutorial is to mobilize the data pertaining to a sales team. In this scenario, each salesperson is a remote synchronization client. Each salesperson has a local SQL Anywhere database that is synchronized to a corporate Oracle database at headquarters using MobiLink. Each salesperson access corporate data with their laptop or handheld device, and manipulates data from the remote database.

Required software

This tutorial assumes that you have a complete install of SQL Anywhere (including MobiLink) on your local computer where you are running the tutorial. This tutorial was created using Oracle Database 10g Release 2. The tutorial may work for other versions of Oracle, but this is not guaranteed. This tutorial also assumes that you have installed Oracle Database 10g on your local computer, but you may also access it remotely.

This tutorial assumes you performed a basic installation of Oracle Database 10g, which creates a starter database named orcl. The orcl database has the Order Entry (OE) and Human Relations (HR) sample schemas. Alternatively, you may manually install the sample schemas or use the Oracle Database Configuration Assistant. For more information about installing both sample schemas, see http://www.oracle.com/technology/obe/obe1013jdev/common/OBEConnection.htm.

This tutorial assumes that you can connect to Oracle as the SYS user with SYSDBA privileges. This is a requirement in Lesson 7 when you grant permission for the GV_$TRANSACTION Oracle system view. The password for the SYS user is set during installation of an Oracle database.

Overview

This tutorial shows you how to:

- Make important considerations, such as synchronization directions for remote tables, when designing a remote schema.
- Add unique primary keys to consolidated and remote databases.
- Create an ODBC data source that connects MobiLink to an Oracle Database 10g.
- Set up synchronization between a consolidated database and remote database using the **Create Synchronization Model Wizard**.
- Customize a synchronization model using Sybase Central.
- Deploy a consolidated database and remote database using the **Deploy Synchronization Model Wizard**.
- Synchronize the remote client with the consolidated database.

Suggested background reading

For an overview of MobiLink synchronization, see "Understanding MobiLink synchronization" on page 1.

For an overview of MobiLink models, see "MobiLink Plug-in for Sybase Central" on page 20.

Lesson 1: Design the schemas

This tutorial assumes that you have installed the Order Entry (OE) and Human Relations (HR) sample schemas. The OE schema is used as the consolidated database. It encapsulates information about employees, orders, customers, and products. For this tutorial, you are primarily interested in the OE schema. However, you must refer to the EMPLOYEES table in the HR schema to get information about each individual salesperson. Here is a brief description of the relevant tables in the OE schema:

Table	Description
CUSTOMERS	Customers whose information is kept on record.
INVENTORIES	How much of each product is stored in each warehouse.
ORDER_ITEMS	List of products included in each order.
ORDERS	Record of a sale between a salesperson and a customer on a specific date.
PRODUCT_DESCRIPTIONS	Descriptions of each product in different languages.
PRODUCT_INFORMATION	A record of each product in the system.

Designing the remote schema

The first step is to design a remote schema. It is unnecessary and inefficient for each salesperson to have a copy of the entire consolidated database. The remote schema is designed so that it only contains

information relevant to one particular salesperson. To achieve this, the remote schema is designed in the following way:

Consolidated table	Remote table
CUSTOMERS	Includes all rows.
INVENTORIES	Not included on remote.
ORDER_ITEMS	Filter by sales_rep_id.
ORDERS	Includes all rows.
PRODUCT_DESCRIPTIONS	Not included on remote.
PRODUCT_INFORMATION	Includes all rows.

Each salesperson needs to keep records of all customers and products, so that any product can be sold to any customer. This tutorial assumes that a salesperson always speaks the same language as the customer, so you do not need the PRODUCT_DESCRIPTIONS table. Each salesperson needs information about orders, but not orders related to other salespeople. This is achieved by filtering rows based on salesperson identifier.

The next step is to choose the synchronization direction of each table. You should consider what information a remote database needs to read and what information a remote database needs to create, change, or remove. In this example, a specific salesperson needs a list of products and customers, but never entered a new product into the system. You are making the restriction that products and customers always enter the system from the consolidated database at headquarters. However, a salesperson needs to be able to record new orders on a regular basis. These factors lead to the following decisions about the synchronization in each table:

Table	Synchronization
CUSTOMERS	Download to remote only.
ORDER_ITEMS	Download and upload.
ORDER	Download and upload.
PRODUCT_INFORMATION	Download to remote only.

Lesson 2: Prepare the consolidated database

This tutorial assumes that you have installed the Order Entry (OE) sample database. Information about installing the sample schema can be found in the Oracle documentation or at http://www.oracle.com/technology/obe/obe1013jdev/common/OBEConnection.htm.

The OE database needs to be altered for use with MobiLink. Columns are dropped because they were created as user-defined types. You could translate these user-defined types into types that SQL Anywhere recognizes, but doing so is not relevant to this tutorial. Next, you must grant permission to the OE user to create triggers because MobiLink needs to create a few triggers using OE's credentials.

To prepare the consolidated database

1. Connect as the SYS user with SYSDBA privileges using the Oracle SQL Plus application. At a command prompt, run the following command:

   ```
   sqlplus SYS/your password for sys as SYSDBA
   ```

2. To drop columns created as user-defined types, run the following commands:

   ```
   ALTER TABLE OE.CUSTOMERS DROP COLUMN CUST_ADDRESS;
   ALTER TABLE OE.CUSTOMERS DROP COLUMN PHONE_NUMBERS;
   ALTER TABLE OE.CUSTOMERS DROP COLUMN CUST_GEO_LOCATION;
   ALTER TABLE OE.PRODUCT_INFORMATION DROP COLUMN WARRANTY_PERIOD;
   ```

3. Unlock the OE user and set the password to sql. Run the following command:

   ```
   ALTER USER OE IDENTIFIED BY sql ACCOUNT UNLOCK;
   ```

4. To allow the OE user to create triggers, run the following command:

   ```
   GRANT CREATE ANY TRIGGER TO OE;
   ```

5. To drop the orders_customer foreign key and create a new foreign key that references the customer_id in the customers table, run the following commands:

   ```
   ALTER TABLE OE.ORDERS DROP CONSTRAINT ORDERS_CUSTOMER_ID_FK;
   ALTER TABLE OE.ORDERS ADD CONSTRAINT ORDERS_CUSTOMER_ID_FK
     FOREIGN KEY (CUSTOMER_ID) REFERENCES OE.CUSTOMERS (CUSTOMER_ID);
   ```

Adding unique primary keys

In a synchronization system, the primary key of a table is the only way to uniquely identify a row in different databases and the only way to detect conflicts. Every table that is being mobilized must have a primary key. The primary key must never be updated. You must also guarantee that a primary key value inserted at one database is not inserted in another database.

There are several ways of generating unique primary keys. For simplicity, the method of composite primary keys is used in this tutorial. This method creates primary keys with multiple columns that are unique across consolidated and remote databases.

To add unique primary keys to the consolidated database

1. At a command prompt, execute the following command:

   ```
   sqlplus SYS/your password for sys as SYSDBA
   ```

2. Values added to the SALES_REP_ID must be in the HR.EMPLOYEES table. The ORDERS_SALES_REP_FK foreign key enforces this rule. Execute the following command to drop the foreign key:

```
ALTER TABLE OE.ORDERS
DROP CONSTRAINT ORDERS_SALES_REP_FK;
```

3. The SALES_REP_ID column cannot be added as a primary key because it contains null values. For this tutorial, replace the null values with a value of 1. Run the following command:

   ```
   UPDATE OE.ORDERS
   SET SALES_REP_ID = 1
   WHERE SALES_REP_ID IS NULL;
   ```

4. The ORDER_ID column is the current primary key of the ORDERS table. To drop the current primary key, run the following command:

   ```
   ALTER TABLE OE.ORDERS
   DROP PRIMARY KEY CASCADE;
   ```

5. The composite primary key consists of the SALES_REP_ID column and the ORDER_ID column. To add the composite primary key, run the following command:

   ```
   ALTER TABLE OE.ORDERS
   ADD CONSTRAINT salesrep_order_pk PRIMARY KEY (sales_rep_id, order_id);
   ```

After running these commands, the MobiLink server should have no trouble connecting to the consolidated database and setting it up for synchronization with any number of remotes.

Unique primary keys across all databases

In Lesson 4, the remote schema is created from the consolidated schema. This means that the remote schema has the same primary keys as the consolidated schema.

Columns were specifically chosen to ensure unique primary keys for all databases. For the ORDERS table, the primary key consists of the SALES_REP_ID and ORDER_ID columns. Any inserted value into the remote sales table must have an unique order number (the SALES_REP_ID value is always the same). This ensures uniqueness in each remote ORDERS table. The primary key in the consolidated ORDERS table prevents conflicts if multiple salespeople upload data. Each upload from a salesperson is unique to another salesperson because their SALES_REP_ID values are different.

Further reading

For more information about compliant RDBMSs, see "MobiLink consolidated databases" [*MobiLink - Server Administration*].

For more information about setting up Oracle as a consolidated database, see "Oracle consolidated database" [*MobiLink - Server Administration*].

For more information about different ways of generating unique primary keys, see "Maintaining unique primary keys" [*MobiLink - Server Administration*].

Lesson 3: Connect with MobiLink

In this lesson, you create an ODBC data source that connects MobiLink to the consolidated database.

To connect MobiLink to the consolidated database

1. Create an ODBC data source.

   You should use the iAnywhere Solutions 12 - Oracle ODBC driver that comes with SQL Anywhere 12. Use the following configuration settings:

ODBC tab fields	Values
Data source name	oracle_cons
User ID	OE
Password	sql
TNS service name	orcl
Procedure returns results	not selected.
Array size	60000

   This tutorial assumes you performed a basic installation of Oracle Database 10g, which creates a starter database named orcl. The Order Entry (OE) schema is automatically installed on orcl. If you installed the OE schema on another database, use the name of the database as the ServiceName value.

2. Click **Test Connection** to test the ODBC connection.

After configuring your ODBC data source, you can use Sybase Central to connect to the Oracle database and create a synchronization model.

Further reading

For more information about recommended ODBC drivers for MobiLink, see http://www.sybase.com/detail?id=1011880.

For more information about configuration options for the iAnywhere Solutions 12 - Oracle ODBC driver, see "iAnywhere Solutions 12 - Oracle ODBC driver" [*MobiLink - Server Administration*].

Lesson 4: Create the synchronization model

In this lesson, you create a synchronization model for your consolidated database.

Connect to the consolidated database by creating a new MobiLink project.

To create a new MobiLink project

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.

2. From the **Tools** menu, choose **MobiLink 12** » **New Project**.

3. In the **Name** field, type **oracle_project**.
4. In the **Location** field, type *C:\mlora*, and then click **Next**.
5. Check the **Add A Consolidated Database To The Project** option.
6. In the **Database Display Name** field, type **oracle_cons**.
7. Click **Edit**. The **Connect To A Generic ODBC Database** window appears.
8. In the **User ID** field, type **OE**.
9. In the **Password** field, type the password for the **sql** account.
10. In the **ODBC Data Source name** field, click **Browse** and select **oracle_cons**.
11. Click **OK**, then click **Save**.
12. Check the **Remember The Password** option, and then click **Next**.
13. Choose the **Create a new model** option, and then click **Next**.
14. Check the **Add a remote schema name to the project** option.
15. Type **oracle_remote_schema** for the remote schema name, and then click **Finish**.

The **Create Synchronization Model Wizard** provides step-by-step instructions for setting up synchronization between the consolidated database and remote database.

To create a synchronization model

1. On the **Welcome** page, type **sync_oracle** in the **What Do You Want To Name The New Synchronization Model** field, and then click **Next**.
2. On the **Primary Key Requirements** page, select all three check boxes (you guarantee unique primary keys in Lesson 2). Click **Next**.
3. Select the **oracle_cons** consolidated database from the list, and then click **Next**.
4. Click **No, Create A New Remote Database Schema**, and then click **Next**.
5. On the **New Remote Database Schema** page, in the **Which Consolidated Database Tables And Columns Do You Want To Have In Your Remote Database** list, select the following tables:
   - CUSTOMERS
   - ORDERS
   - ORDER_ITEMS
   - PRODUCT_INFORMATION
6. Click **Next**.
7. Click **Timestamp-based Download**, and then click **Next**.

Timestamp-based downloads minimize the amount of data that is transferred because only data that has been updated since the last download is transmitted.

8. On the **Timestamp Download Options** page, click **Use Shadow Tables To Hold Timestamp Columns**, and then click **Next**.

   Using shadow tables is often preferred because it does not require any changes to existing tables.

9. Perform the following tasks on the **Download Deletes** page:
   a. Click **Yes** on the **Do You Want Data Deleted On The Consolidated Database To Be Deleted On The Remote Databases** option.
   b. Click **Use Shadow Tables To Record Deletions**.

      MobiLink creates shadow tables on the consolidated database to implement deletions.
   c. Click **Next**.

10. Click **Yes, Download the Same Data to Each Remote**, and then click **Next**.

    You specify how to download specific data to a remote database by using custom logic when editing the synchronization model.

11. Click **No Conflict Detection**, and then click **Next**.

    Although this tutorial specifies no conflict detection, many applications require conflict detection.

12. Perform the following tasks on the **Publication, Script Version and Description** page:
    a. In the **What Do You want To Name The Publication** field, type **sync_oracle_publication**.
    b. In the **What Do You Want To Name The Script Version** field, type **sync_oracle_scriptversion**.

       The publication is the object on the remote database that specifies what data is synchronized. MobiLink server scripts define how uploaded data from remotes should be applied to the consolidated database, and script versions group scripts. You can use different script versions for different applications, allowing you to maintain a single MobiLink server while synchronizing different applications.
    c. Click **Finish**.

To edit a synchronization model

1. From the **View** menu, choose **Folders**.
2. In the left pane of Sybase Central under **MobiLink 12**, expand **oracle_project**, **Synchronization Models** and then **sync_oracle**.
3. Set the direction that data is synchronized for each table in the synchronization model.

Click the **Mappings** tab in the right pane, and then set the rows the **Dir** column as follows:

- The **ORDERS** and **ORDER_ITEMS** tables should be set to **Bi-directional** (both upload and download).
- The remaining tables should be set to **Download To Remote Only**.

4. Filter the rows downloaded to the remote database by remote ID.
   a. For the row containing the **ORDERS** table, change the **Download Subset** to **Custom**.
   b. Click the **Download Subset** tab at the bottom of the right pane.
   c. Filter the rows by remote ID, which uniquely identifies the remote database, by adding a restriction to the WHERE clause of the download_cursor script.

      Type a search condition in the **SQL Expression To Use In The Download Cursor's WHERE Clause** field. For example, the following SQL script can be used for the **ORDERS** table:

      ```
      "OE"."ORDERS"."SALES_REP_ID" = {ml s.remote_id}
      ```

      The download cursor script specifies what columns and rows are downloaded from each table to the remote database. The search condition ensures that you only download information about one store, namely, the store that has an identifier that equals the remote ID for the database.

5. Save the synchronization model.

   From the **File** menu, choose **Save**.

   The synchronization model is complete and ready for deployment.

Further reading

- "Setting up a consolidated database" [*MobiLink - Server Administration*]
- "MobiLink server system tables" [*MobiLink - Server Administration*]
- "MobiLink server system procedures" [*MobiLink - Server Administration*]
- "Writing download_delete_cursor scripts" [*MobiLink - Server Administration*]
- "Handling conflicts" [*MobiLink - Server Administration*]
- "Resolving conflicts" [*MobiLink - Server Administration*]
- "Publishing data" [*MobiLink - Client Administration*]
- "Synchronization model tasks" on page 26
- "Modifying the download type" on page 29
- "Modifying conflict detection and resolution" on page 34
- "Modifying table and column mappings" on page 27

Lesson 5: Deploy the synchronization model

The **Deploy Synchronization Model Wizard** allows you to deploy the consolidated database and remote database. You can deploy each of these individually or you can deploy both of them. The **Deploy Synchronization Model Wizard** takes you through the steps of configuring options for deployment.

To deploy the synchronization model

1. In the left pane of Sybase Central under **MobiLink 12**, expand **oracle_project**, **Synchronization Models** and then **sync_oracle**.
2. From the **File** menu, choose **Deploy**.
3. Click **Specify The Deployment Details For One Or More Of The Following** and select **Consolidated Database**, **Remote Database And Synchronization Client**, and **MobiLink Server**. Click **Next**.
4. Perform the following tasks on the **Consolidated Database Deployment Destination** page:
   a. Check **Save Changes To The Following SQL File** and accept the default location for the file.

      MobiLink generates a *.sql* file that makes changes to the consolidated database to set up for synchronization. You can examine the *.sql* file later and make your own changes. Then, you must run the *.sql* file yourself.

   b. Immediately apply the changes to the consolidated database.

      Check **Connect To The Consolidated Database To Directly Apply The Changes**.

   c. Select the **oracle_cons** consolidated database from the list .
   d. Click **Next**.

   A prompt appears asking if you want to create the *consolidated* directory. Click **Yes**.
5. Click **New SQL Anywhere Database**, and then click **Next**.
6. Perform the following tasks on the **New SQL Anywhere Remote Database** page:
   a. Check the **Make A Command File And A SQL File With Commands To Create The Database** option.

      Selecting this option generates another *.sql* file with the commands to set up the remote database with all schema and synchronization information.

   b. Accept the default location in the **SQL File** field.
   c. Check the **Create A Remote SQL Anywhere Database** option.

      You must generate a new remote database and then run the *.sql* file against it If you do not check this option This option allows you to examine the *.sql* file later and make your own changes.

   d. Accept the default location in the **SQL Anywhere Database File** field.
   e. Click **Next**.

   A prompt appears asking if you want to create the *remote* directory. Click **Yes**.
7. Perform the following tasks on the **MobiLink User and Subscription** page:
   a. In the **What User Name Do You Want To Use For Connecting To The MobiLink server** field, type **oracle_remote**.
   b. In the **What Password Do You Want To Use** field, type **oracle_pass**.

c. Click **Next**.

8. Select **TCP/IP** and type **2439** in the **Port** field. Click **Next**.

9. Type **localhost** in the **Host** field. Click **Next**.

   Alternatively, you can type your computer name or IP address, the name or IP address of another network server you want to use, or other client stream options.

10. Click **Next** on the **Client Stream Parameters**, **MobiLink Server Stream Parameters** and **Verbosity For MobiLink Server** pages to accept all default settings.

11. Type **oracle_mlsrv** in the **What Name Do You Want To Give The MobiLink Server** field. Click **Next**.

    A message appears asking if you want to create the *mlsrv* directory. Click **Yes**.

12. Choose a verbosity setting for the remote synchronization client and use the default file name of the remote database log file. Click **Next**.

13. Click **Finish**.

14. Click **Close**.

Your consolidated database is fully configured for synchronization with many remote clients, and you have successfully deployed one remote client. If you want to deploy other remote clients, you can run this wizard again, making sure to create a new MobiLink user and opt out of deploying the consolidated database and MobiLink server. Since these have already been deployed, all you need to do is deploy other remote synchronization clients.

Further reading

- "Deploying synchronization models" on page 38
- "Creating a remote database" [*MobiLink - Client Administration*]
- "Introduction to MobiLink users" [*MobiLink - Client Administration*]

Lesson 6: Start the server and client

In this lesson, you start the MobiLink server and remote database.

Previously, you modified the download cursor script to download information related to one salesperson. In this lesson, you specify the salesperson by setting the remote ID to the salesperson identifier.

Start the MobiLink server

By default, MobiLink uses the snapshot/READ COMMITTED isolation level for upload and download. For the MobiLink server to make the most effective use of snapshot isolation, the Oracle account used by the MobiLink server must have access to the GV_$TRANSACTION Oracle system view. If access is not given, a warning is issued and rows may be missed on download.

To grant access to the OE user

1. Connect as the SYS user with SYSDBA privileges using the Oracle SQL Plus application. At a command prompt, run the following command:

   ```
   sqlplus SYS/your password for sys as SYSDBA
   ```

2. To grant access to the GV_$TRANSACTION Oracle system view, run the following command:

   ```
   GRANT SELECT ON SYS.GV_$TRANSACTION TO OE;
   ```

3. To grant access to the V_$SESSION Oracle system view, run the following command:

   ```
   GRANT SELECT ON SYS.V_$SESSION TO OE;
   ```

To start the MobiLink server

1. At a command prompt, navigate to the directory where you created the synchronization model. (This is the root directory you chose in the first step of the **Create Synchronization Model Wizard**.)

   If you used the suggested directory names, you should have the following directories located in the root directory: *sync_oracle\mlsrv*.

2. Run the following command from the mlsrv directory:

   ```
   sync_oracle_mlsrv.bat "DSN=oracle_cons;UID=OE;PWD=sql;"
   ```

   - **sync_oracle_mlsrv.bat** is the command file created to start the MobiLink server.
   - **DSN** is your ODBC data source name.
   - **UID** is the user name you use to connect to the consolidated database.
   - **PWD** is the password you use to connect to the consolidated database.

   When this command runs successfully, the message `MobiLink server Started` appears in the MobiLink server messages window.

   If the MobiLink server fails to start, check your connection information for your consolidated database.

To start the remote database

1. At a command prompt, navigate to the directory where the **Deploy Synchronization Model Wizard** created your remote database.

   If you used the suggested directory names, you should have the following directories located in the root directory: *sync_oracle\remote*.

2. Start your remote SQL Anywhere database with the following command:

   ```
   dbeng12 -n remote_eng sync_oracle_remote.db -n remote_db
   ```

   - **dbeng12** is the database server used to start the SQL Anywhere database.
   - **remote_eng** is the database server name.

- **sync_oracle_remote.db** is the database file that is started on remote_eng.
- **remote_db** is the name of the database on remote_eng.

When this command runs successfully, a SQL Anywhere database server named remote_eng starts and loads the database called remote_db.

Set the remote ID

In the remote schema, each remote database represents one salesperson. The synchronization scripts you wrote included logic that instructed the MobiLink server to download a subset of data based on the remote ID of the remote database. You must set the database's remote ID to the value of a valid salesperson identifier.

It is important to complete this step before the first synchronization because when the remote device synchronizes for the first time, it downloads all information related to the chosen salesperson.

To set the remote ID to a valid salesperson identifier

1. Choose a valid salesperson identifier:
   a. Connect as the SYS user with SYSDBA privileges using the Oracle SQL Plus application. At a command prompt, run the following command:

      ```
      sqlplus SYS/your password for sys as SYSDBA
      ```

   b. To view a list of valid salesperson identifiers in the ORDERS table, execute the following statement:

      ```
      SELECT COUNT( SALES_REP_ID ), SALES_REP_ID
      FROM OE.ORDERS GROUP BY SALES_REP_ID;
      ```

      In this example, the remote database represents a salesperson with a SALES_REP_ID of 154.

   c. To exit Oracle, run the following command:

      ```
      exit
      ```

2. To set the database's remote id to a value of 154, run the following command:

   ```
   dbisql
   -c "server=remote_eng;DBN=remote_db;UID=DBA;PWD=sql"
   "SET OPTION PUBLIC.ml_remote_id='154'"
   ```

   - **dbisql** is the application used to execute SQL commands against a SQL Anywhere database.
   - **ENG** specifies the database server name remote_eng.
   - **DBN** specifies the database name to remote_db.
   - **UID** is the user name used to connect to your remote database.
   - **PWD** is the password used to connect to your remote database.
   - **SET OPTION PUBLIC.ml_remote_id='154'** is the SQL command used to set the remote ID to a value of 154.

Further reading

- "The SQL Anywhere database server" [*SQL Anywhere Server - Database Administration*]
- "Synchronizing a deployed model" on page 41
- "Running the MobiLink server" [*MobiLink - Server Administration*]
- "Remote IDs" [*MobiLink - Client Administration*]

Lesson 7: Synchronize

Now you are ready to synchronize the remote client for the first time. This is done with the MobiLink client program dbmlsync. Dbmlsync connects to the remote database, authenticates itself with the MobiLink server, and performs all the uploads and downloads necessary to synchronize the remote and consolidated databases based on a publication in the remote database.

To synchronize the remote client

- At a command prompt, run the following command:

```
dbmlsync
-c "server=remote_eng;DBN=remote_db;UID=DBA;PWD=sql;"
-n sync_oracle_publication
-u oracle_remote -mp oracle_pass
```

  - **dbmlsync** is the synchronization application.
  - **ENG** specifies the name of the remote database server.
  - **DBN** specifies the name of the remote database.
  - **UID** is the user name used to connect to the remote database.
  - **PWD** is the password used to connect to the remote database.
  - **sync_oracle_publication** is the publication on the remote device that is used to perform the synchronization. (This publication was created by the **Create Synchronization Model Wizard**.)
  - **oracle_remote** is the user name used to authenticate with the MobiLink server.
  - **oracle_pass** is the password used to authenticate with the MobiLink server.

The progress of the synchronization appears in the **SQL Anywhere MobiLink Client Messages** window. When this command runs successfully, the dbmlsync application populates the remote database with a subset of information from the consolidated database.

If synchronization fails, check the connection information you pass to the dbmlsync application, and the MobiLink user name and password. Failing that, check the publication name you used, and ensure that the consolidated database and MobiLink server are running. You can also examine the contents of the synchronization logs (server and client).

Note
If you are running the dbmlsync application on a different computer from your MobiLink server, you must pass in arguments that specify the location of the MobiLink server.

View the data

After successfully synchronizing the remote client to the consolidated database through the MobiLink server, the remote data should be populated with information relevant to one salesperson. You can verify this in Sybase Central using the SQL Anywhere 12 plug-in.

To view the data in the remote database

1. Start Sybase Central.
2. Connect to the remote database:
   a. On the left pane, right-click **SQL Anywhere 12** and choose **Connect**.
   b. In the **Authentication** dropdown list, choose **Database** and type **DBA** as the **User Id** and **sql** as the **Password**.
   c. From the **Action** dropdown list, choose **Connect To A Running Database On This Computer**, and then type **remote_eng** as the **Server Name** and **remote_db** as the **Database Name**.
   d. Click **OK**.
3. Choose the ORDERS table and then click the **Data** tab on the right pane.

   In the ORDERS tables, all the records are for the salesperson with an identifier of 154. This particular salesperson is not concerned with the sales information of other salespeople. For this reason, you set the synchronization scripts to filter out rows by the remote ID, and you set this database's remote ID to the value of a particular salesperson identifier. Now this particular salesperson's database takes up less space, and requires less time to synchronize. Since the remote database size is kept to a minimum, frequently performed operations such as entering a new sale or processing a refund on a previous sale runs faster and more efficiently.

Further reading

- "The synchronization process" on page 13
- "dbmlsync syntax" [*MobiLink - Client Administration*]

Cleanup

Regenerate the Order Entry database and remove all tutorial materials from your computer.

To regenerate the Order Entry database

- Run *oe_main.sql* to remove the current OE schema and install a new OE schema. The *oe_main.sql* file is found in *$ORACLE_HOME/demo/schema/order_entry*.

Alternatively, you may use the Oracle Database Configuration Assistant to create a new database with newly installed sample schemas.

To delete the synchronization model

1. Start Sybase Central.

2. In the right pane, double-click **MobiLink 12**.

   The **sync_oracle** model appears in the right pane.

3. Right click **sync_oracle** and click **Delete**.

4. In the **Confirm Delete** window, click **Delete**.

To erase the remote database

- To erase the remote database, use the dberase utility. Run the following command:

```
dberase sync_oracle\remote\sync_oracle_remote.db
```

Further reading

For more information about running the MobiLink server, see "MobiLink server" [*MobiLink - Server Administration*].

For more information about writing synchronization scripts, see "Writing synchronization scripts" [*MobiLink - Server Administration*] and "Synchronization events" [*MobiLink - Server Administration*].

For an introduction to other methods of synchronization, such as timestamp, see "Synchronization techniques" [*MobiLink - Server Administration*].

Tutorial: Using MobiLink with an Adaptive Server Enterprise consolidated database

Introduction to Adaptive Server Enterprise tutorial

This tutorial shows you how to mobilize an Adaptive Server Enterprise database using MobiLink. It sets up synchronization between an Adaptive Server Enterprise consolidated database and a SQL Anywhere remote database. You could also use UltraLite clients.

The purpose of this tutorial is to mobilize data for a chain of bookstores. Each bookstore in this scenario is a remote synchronization environment. Each bookstore has a local SQL Anywhere database that is synchronized with the Adaptive Server Enterprise database at headquarters. Each bookstore can have several computers that access and manipulate data from the remote database.

Required software

This tutorial assumes you have a complete install of SQL Anywhere (including MobiLink) on your local computer where you are running the tutorial. This tutorial was created using Adaptive Server Enterprise 15.0. The tutorial may work for other versions of Adaptive Server Enterprise, but this is not guaranteed. This tutorial also assumes you have installed Adaptive Server Enterprise 15.0 on your local computer. You may also access Adaptive Server Enterprise 15.0 remotely using Sybase Open Client.

This tutorial assumes that the pubs2 sample schema is installed on an Adaptive Server Enterprise server. The pubs2 sample schema is provided with Adaptive Server Enterprise 15.0 and it is an optional part of the install. For this tutorial, it is used as the consolidated database. Information about this sample can be found in the Adaptive Server Enterprise documentation or at http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.ase_15.0.sqlug/html/sqlug/sqlug894.htm.

This tutorial uses the default **sa** account. When Adaptive Server Enterprise is installed, the **sa** account has a null password. This tutorial assumes you have changed the null password to a valid password. For more information about changing the null password in Adaptive Server Enterprise, see http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.ase_15.0.sag1/html/sag1/sag1615.htm.

Overview

This tutorial shows you how to:

- Make important considerations, such as synchronization directions for remote tables, when designing a remote schema.
- Add unique primary keys to consolidated and remote databases.
- Create an ODBC data source that connects MobiLink to an Adaptive Server Enterprise database.
- Set up synchronization between a consolidated database and remote database using the **Create Synchronization Model Wizard**.
- Customize synchronization settings using Sybase Central.
- Deploy a consolidated database and remote database using the **Deploy Synchronization Model Wizard**.
- Synchronize the remote client with the consolidated database.

Suggested background reading

For an overview of MobiLink synchronization, see "Understanding MobiLink synchronization" on page 1.

Lesson 1: Design your schemas

This tutorial assumes that the pubs2 sample schema is installed on an Adaptive Server Enterprise server. The Adaptive Server Enterprise server may be installed locally on your computer or accessed remotely using Sybase Open Client.

The pubs2 sample schema is used as the consolidated database schema. It contains information about stores, titles, authors, publishers, and sales. The following table provides a description of each table in the Adaptive Server Enterprise database:

Table	Description
au_pix	Pictures of the authors.

Table	Description
authors	The authors of the various TITLES in the system.
discounts	Records of various discounts at particular STORES.
sales	Each sale record is one sale made by a particular store.
salesdetail	Contains information about the different TITLES that were included in a particular sale.
stores	Each store record is one store or branch office in the system.
titleauthor	Contains information about which TITLES were written by which AUTHORS.
titles	Records of all the different books in the system.
blurbs, publishers, and roysched	Contains information that is not needed in this demonstration.

Designing the remote schema

It is unnecessary and inefficient for each store to have a copy of the entire consolidated database. The remote schema uses the same table names, but only contains information relevant to one particular store. To achieve this, the remote schema is designed as a subset of the consolidated database in the following way:

Consolidated table	Remote table
au_pix	Includes all rows.
authors	Includes all rows.
discounts	Filter by stor_id.
sales	Filter by stor_id.
salesdetail	Filter by stor_id.
stores	Filter by stor_id.
titleauthor	Includes all rows.
titles	Includes all rows.
blurbs	Not included on remote.
publishers	Not included on remote.

Consolidated table	Remote table
roysched	Not included on remote.

Each store needs to keep records of all titles and authors so customers can search their inventory. However, a bookstore does not need information about publishers or royalties, so this information is not synchronized to each store. Each store needs information about sales and discounts, but not about sales and discounts related to other stores. This is achieved by filtering rows based on a store identifier.

Note
You can also take a subset of columns from a table if certain columns are not required on the remote databases.

The next step is to choose the synchronization direction of each table. You should consider what information a remote database needs to read and what information a remote database needs to create, change, or remove. In this example, a bookstore needs access to the list of authors and titles, but never enters a new author into the system. This places a restriction that authors and titles must always enter the system from the consolidated database at headquarters. However, a bookstore needs to be able to record new sales on a regular basis. These factors lead to the following synchronization directions for the tables:

Table	Synchronization
titleauthor	Download to remote only.
authors	Download to remote only.
au_pix	Download to remote only.
titles	Download to remote only.
stores	Download to remote only.
discounts	Download to remote only.
sales	Download and upload.
salesdetail	Download and upload.

Lesson 2: Prepare the consolidated database

When connecting to the pubs2 database, this tutorial uses the default **sa** account. When Adaptive Server Enterprise is installed, the **sa** account has a null password. This tutorial assumes you have changed the null password to a valid password. For more information about changing the null password in Adaptive Server Enterprise, see http://infocenter.sybase.com/help/index.jsp?topic=/com.sybase.help.ase_15.0.sag1/html/sag1/sag1615.htm.

In this lesson, you increase the size of the consolidated database for MobiLink synchronization and create unique primary keys.

Increasing the size of the consolidated database

MobiLink needs to add system tables and other objects to the pubs2 database for synchronization. To do this, the size of the pubs2 database must be increased.

To increase the size of the consolidated database

1. Connect to the pubs2 database as **sa**, using the isql utility in Adaptive Server Enterprise. At a command prompt, run the following command, all on one line:

   ```
   isql
   -U sa
   -P Your password for sa account
   -D pubs2
   ```

   If you are accessing Adaptive Server Enterprise remotely, use the -S parameter to specify the server name.

2. To have proper permission for increasing the size of a database, you must access the master database. Run the following command in isql:

   ```
   use master
   ```

3. In Adaptive Server Enterprise, a database is stored on a disk or a portion of a disk. To increase the pubs2 database, run the following command (you must specify the disk where pubs2 is stored):

   ```
   ALTER DATABASE pubs2 ON disk name = 33
   ```

Adding unique primary keys

In a synchronization system, the primary key of a table is the only way to uniquely identify a row in different databases and the only way to detect conflicts. Every table that is being mobilized must have a primary key. The primary key must never be updated. You must also guarantee that a primary key value inserted at one database is not inserted in another database.

There are several ways to generate unique primary keys. For simplicity, the method of composite primary keys is used in this tutorial. This method creates primary keys with multiple columns that are unique across consolidated and remote databases.

To add unique primary keys to the consolidated database

1. Connect to the pubs2 database as **sa**, using the isql utility in Adaptive Server Enterprise. At a command prompt, run the following command, all on one line:

   ```
   isql
   -U sa
   -P Your password for sa account
   -D pubs2
   ```

   If you are accessing Adaptive Server Enterprise remotely, use the -S parameter to specify the server name.

2. The following rows are not unique based on the composite primary key created for the salesdetail table in step 5. For simplicity, drop the rows by running the following commands:

```
DELETE FROM salesdetail
WHERE stor_id = '5023'
AND ord_num = 'NF-123-ADS-642-9G3'
AND title_id = 'PC8888'

DELETE FROM salesdetail
WHERE stor_id = '5023'
AND ord_num = 'ZS-645-CAT-415-1B2'
AND title_id = 'BU2075'
```

3. The following indexes interfere with the creation of primary keys in step 5. To drop the indexes, run the following commands:

```
DROP INDEX authors.auidind
DROP INDEX titleauthor.taind
DROP INDEX titles.titleidind
DROP INDEX sales.salesind
```

4. Add unique primary keys.

```
ALTER TABLE au_pix ADD PRIMARY KEY (au_id)
ALTER TABLE authors ADD PRIMARY KEY (au_id)
ALTER TABLE titleauthor ADD PRIMARY KEY (au_id, title_id)
ALTER TABLE titles ADD PRIMARY KEY (title_id)
ALTER TABLE discounts ADD PRIMARY KEY (discounttype)
ALTER TABLE stores ADD PRIMARY KEY (stor_id)
ALTER TABLE sales ADD PRIMARY KEY (stor_id, ord_num)
ALTER TABLE salesdetail ADD PRIMARY KEY (stor_id, ord_num, title_id)
```

After running these commands, the MobiLink server should have no trouble connecting to the consolidated database and setting it up for synchronization with any number of remotes.

Note
It is possible to synchronize data with consolidated databases that do not have primary keys. However, you must write you own synchronization events that act on shadow tables that are designed to identify rows uniquely in other tables.

Unique primary keys across all databases

In Lesson 4, the remote schema is created from the consolidated schema. This means that the remote schema has the same primary keys as the consolidated schema.

Columns were specifically chosen to ensure unique primary keys for all databases. For the sales table, the primary key consists of the stor_id and ord_num columns. Any inserted value into the remote sales table must have an unique order number (the stor_id value is always the same). This ensures uniqueness in each remote sales table. The primary key in the consolidated sales table prevents conflicts if multiple stores upload data. Each upload from one store is unique to another store because their stor_id values are different.

For the salesdetail table, the primary key consists of the stor_id, ord_num and title_id columns. There may be multiple book titles in an order. For the remote sales tables, rows may have the same values for stor_id and ord_num, but they must have different title_id values. This ensures uniqueness in each remote salesdetail table. Similar to sales table, each upload to the consolidated database from a store is unique to another store because their stor_id values are different.

Further reading

For more information about Adaptive Server Enterprise issues, see "Adaptive Server Enterprise consolidated database" [*MobiLink - Server Administration*].

For more information about different ways of generating unique primary keys, see "Maintaining unique primary keys" [*MobiLink - Server Administration*].

Lesson 3: Connect with MobiLink

In this lesson, you create an ODBC data source that connects MobiLink to the consolidated database.

To connect MobiLink to the consolidated database

1. Create an ODBC data source.

   You should use the ODBC driver provided by Adaptive Server Enterprise. For this tutorial, use the following configuration settings:

General tab fields	Value
Data Source Name	ase_cons
Description	
Server Name (ASE Host Name)	localhost
Server Port	5000
Database Name	pubs2
Logon ID	sa
Use Cursors	not selected

Transaction tab field	Value
Server Initiated Transactions	not selected

2. Test the ODBC connection:
   a. On the **General** tab, click **Test Connection**.

      The Adaptive Server Enterprise logon screen appears.

   b. Enter the password for the **sa** account.

      The **Logon Succeeded** message appears.

After configuring your ODBC data source, you can use the MobiLink 12 plug-in to connect to the consolidated database and create a synchronization model.

Further reading

For more information about recommended ODBC drivers for MobiLink, see http://www.sybase.com/detail?id=1011880.

Lesson 4: Create a synchronization model

In this lesson, you create a synchronization model for your consolidated database.

Connect to the consolidated database by creating a new MobiLink project.

To create a new MobiLink project

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.
2. From the **Tools** menu, choose **MobiLink 12** » **New Project**.
3. In the **Name** field, type **ase_project**.
4. In the **Location** field, type *C:\mlase*, and then click **Next**.
5. Check the **Add A Consolidated Database To The Project** option.
6. In the **Database Display Name** field, type **ase_cons**.
7. Click **Edit**. The **Connect To A Generic ODBC Database** window appears.
8. In the **User ID** field, type **sa**.
9. In the **Password** field, type the password for the **sa** account.
10. In the **ODBC Data Source name** field, click **Browse** and select **ase_cons**.
11. Click **OK**, then click **Save**.
12. Check the **Remember The Password** option, and then click **Next**.
13. Choose the **Create a new model** option, and then click **Next**.
14. Check the **Add a remote schema name to the project** option.
15. Type **ase_remote_schema** for the remote schema name, and then click **Finish**.

The **Create Synchronization Model Wizard** provides step-by-step instructions for setting up synchronization between the consolidated database and remote database.

To create a synchronization model

1. On the **Welcome** page, type **sync_ase** in the **What Do You Want To Name The New Synchronization Model** field, and then click **Next**.

2. On the **Primary Key Requirements** page, select all three check boxes (you guarantee unique primary keys in Lesson 2). Click **Next**.

3. Select the **ase_cons** consolidated database from the list, and then click **Next**.

4. Click **No, Create A New Remote Database Schema**, and then click **Next**.

5. On the **New Remote Database Schema** page, in the **Which Consolidated Database Tables And Columns Do You Want To Have In Your Remote Database** list, select the following tables:
   - au_pix
   - authors
   - discounts
   - sales
   - salesdetail
   - stores
   - titleauthor
   - titles

6. Click **Next**.

7. Click **Timestamp-based Download**, and then click **Next**.

   Timestamp-based downloads minimize the amount of data that is transferred because only data that has been updated since the last download is transmitted.

8. On the **Timestamp Download Options** page, click **Use Shadow Tables To Hold Timestamp Columns**, and then click **Next**.

   Using shadow tables is often preferred because it does not require any changes to existing tables.

9. Perform the following tasks on the **Download Deletes** page:
   a. Click **Yes** on the **Do You Want Data Deleted On The Consolidated Database To Be Deleted On The Remote Databases** option.
   b. Click **Use Shadow Tables To Record Deletions**.

      MobiLink creates shadow tables on the consolidated database to implement deletions.
   c. Click **Next**.

10. Click **Yes, Download the Same Data to Each Remote**, and then click **Next**.

    You specify how to download specific data to a remote database by using custom logic when editing the synchronization model.

11. Click **No Conflict Detection**, and then click **Next**.

    Although this tutorial specifies no conflict detection, many applications require conflict detection.

12. Perform the following tasks on the **Publication, Script Version and Description** page:

a. In the **What Do You want To Name The Publication** field, type **sync_ase_publication**.

b. In the **What Do You Want To Name The Script Version** field, type **sync_ase_scriptversion**.

The publication is the object on the remote database that specifies what data is synchronized. MobiLink server scripts define how uploaded data from remotes should be applied to the consolidated database, and script versions group scripts. You can use different script versions for different applications, allowing you to maintain a single MobiLink server while synchronizing different applications.

c. Click **Finish**.

To edit a synchronization model

1. From the **View** menu, choose **Folders**.
2. In the left pane of Sybase Central under **MobiLink 12**, expand **ase_project**, **Synchronization Models** and then **sync_ase**.
3. Set the direction that data is synchronized for each table in the synchronization model.

   Click the **Mappings** tab in the right pane, and then set the rows the **Dir** column as follows:

   - The **sales** and **salesdetail** tables should be set to **Bi-directional** (both upload and download).
   - The remaining tables should be set to **Download To Remote Only**.
4. Filter the rows downloaded to the remote database by remote ID.

   a. For the row containing the **stores** table, change the **Download Subset** to **Custom**.

   b. Click the **Download Subset** tab at the bottom of the right pane.

   c. Filter the rows by remote ID, which uniquely identifies the remote database, by adding a restriction to the WHERE clause of the download_cursor script.

   Type a search condition in the **SQL Expression To Use In The Download Cursor's WHERE Clause** field. For example, the following SQL script can be used for the **stores** table:

   ```
   "dbo"."stores"."stor_id" = {ml s.remote_id}
   ```

   The download cursor script specifies what columns and rows are downloaded from each table to the remote database. The search condition ensures that you only download information about one store, namely, the store that has an identifier that equals the remote ID for the database.
5. Repeat step 3 for the rows containing the **sales**, **salesdetail**, and **discounts** tables.

Note
You must rename the table specified in the SQL script to the table name in the row that you are editing.

Use the following WHERE clause script for the **salesdetail** table:

```
"dbo"."salesdetail"."stor_id" = {ml s.remote_id}
```

Use the following WHERE clause script for the **discounts** table:

```
"dbo"."discounts"."stor_id" = {ml s.remote_id}
```

6. Save the synchronization model.

   From the **File** menu, choose **Save**.

   The synchronization model is complete and ready for deployment.

Further reading

- "Setting up a consolidated database" [*MobiLink - Server Administration*]
- "MobiLink server system tables" [*MobiLink - Server Administration*]
- "MobiLink server system procedures" [*MobiLink - Server Administration*]
- "Writing download_delete_cursor scripts" [*MobiLink - Server Administration*]
- "Handling conflicts" [*MobiLink - Server Administration*]
- "Resolving conflicts" [*MobiLink - Server Administration*]
- "Publishing data" [*MobiLink - Client Administration*]
- "Synchronization model tasks" on page 26
- "Modifying the download type" on page 29
- "Modifying conflict detection and resolution" on page 34
- "Modifying table and column mappings" on page 27

Lesson 5: Deploy the synchronization model

The **Deploy Synchronization Model Wizard** allows you to deploy the consolidated database and remote database. You can deploy each of these individually or you can deploy both of them. The **Deploy Synchronization Model Wizard** takes you through the steps of configuring options for deployment.

To deploy the synchronization model

1. In the left pane of Sybase Central under **MobiLink 12**, expand **ase_project**, **Synchronization Models** and then **sync_ase**.

2. From the **File** menu, choose **Deploy**.

3. Click **Specify The Deployment Details For One Or More Of The Following** and select **Consolidated Database**, **Remote Database And Synchronization Client**, and **MobiLink Server**. Click **Next**.

4. Perform the following tasks on the **Consolidated Database Deployment Destination** page:

a. Check **Save Changes To The Following SQL File** and accept the default location for the file.

   MobiLink generates a *.sql* file that makes changes to the consolidated database to set up for synchronization. You can examine the *.sql* file later and make your own changes. Then, you must run the *.sql* file yourself.

b. Immediately apply the changes to the consolidated database.

   Check **Connect To The Consolidated Database To Directly Apply The Changes**.

c. Select the **ase_cons** consolidated database from the list .

d. Click **Next**.

A prompt appears asking if you want to create the *consolidated* directory. Click **Yes**.

5. Click **New SQL Anywhere Database**, and then click **Next**.

6. Perform the following tasks on the **New SQL Anywhere Remote Database** page:

   a. Check the **Make A Command File And A SQL File With Commands To Create The Database** option.

      Selecting this option generates another *.sql* file with the commands to set up the remote database with all schema and synchronization information.

   b. Accept the default location in the **SQL File** field.

   c. Check the **Create A Remote SQL Anywhere Database** option.

      You must generate a new remote database and then run the *.sql* file against it If you do not check this option This option allows you to examine the *.sql* file later and make your own changes.

   d. Accept the default location in the **SQL Anywhere Database File** field.

   e. Click **Next**.

   A prompt appears asking if you want to create the *remote* directory. Click **Yes**.

7. Perform the following tasks on the **MobiLink User and Subscription** page:

   a. In the **What User Name Do You Want To Use For Connecting To The MobiLink Server** field, type **ase_remote**.

   b. In the **What Password Do You Want To Use** field, type **ase_pass**.

   c. Click **Next**.

8. Select **TCP/IP** and type **2439** in the **Port** field. Click **Next**.

9. Type **localhost** in the **Host** field. Click **Next**.

   Alternatively, you can type your computer name or IP address, the name or IP address of another network server you want to use, or other client stream options.

10. Click **Next** on the **Client Stream Parameters**, **MobiLink Server Stream Parameters** and **Verbosity For MobiLink Server** pages to accept all default settings.

11. Type **ase_mlsrv** in the **What Name Do You Want To Give The MobiLink Server** field. Click **Next**.

    A message appears asking if you want to create the *mlsrv* directory. Click **Yes**.

12. Choose a verbosity setting for the remote synchronization client and use the default file name of the remote database log file. Click **Next**.

13. Click **Finish**.

14. Click **Close**.

Your consolidated database is fully configured for synchronization with many remote clients, and you have successfully deployed one remote client. If you want to deploy other remote clients, you can run this wizard again, making sure to create a new MobiLink user and opt out of deploying the consolidated database and MobiLink server. Since these have already been deployed, all you need to do is deploy other remote synchronization clients.

Further reading

- "Deploying synchronization models" on page 38
- "Creating a remote database" [*MobiLink - Client Administration*]
- "Introduction to MobiLink users" [*MobiLink - Client Administration*]

Lesson 6: Start the server and client

In this lesson, you start the MobiLink server and remote database.

Previously, you modified the download cursor script to download information related to one store. In this lesson, you specify the store by setting the remote ID to the store identifier.

To start the MobiLink server

1. At a command prompt, navigate to the folder where you created the synchronization model. (This is the root directory you chose in the first step of the **Synchronization Model Wizard**.)

   If you used the suggested directory names, you should navigate to the following directory: *sync_ase \mlsrv*.

2. To start the MobiLink server, run the following command:

   ```
   sync_ase_mlsrv.bat "dsn=ase_cons;uid=sa;pwd=sa;"
   ```

   - **sync_ase_mlsrv.bat** is the command file to start the MobiLink server.
   - **dsn** is your ODBC data source name.
   - **uid** is the user name you use to connect to the consolidated database (the default for Adaptive Server Enterprise is **sa**).
   - **pwd** is the password you use to connect as **sa**.

   When this command runs successfully, the message `MobiLink server Started` appears in the MobiLink server messages window.

If the MobiLink server fails to start, check connection information for your consolidated database.

To start the remote database

1. At a command prompt, navigate to the directory where the **Deploy Synchronization Model Wizard** created your remote database.

   If you used the suggested directory names, you navigate to the following directory: *sync_ase\remote*.

2. To start your remote SQL Anywhere database, run the following command:

   ```
   dbeng12 -n remote_eng sync_ase_remote.db -n remote_db
   ```

   - **dbeng12** is the database server used to start the SQL Anywhere database.
   - **remote_eng** is the database server name.
   - **sync_ase_remote.db** is the database file that is started on remote_eng.
   - **remote_db** is the name of the database on remote_eng.

   When this command runs successfully, a SQL Anywhere database server named remote_eng starts and loads the database called remote_db.

Set the remote ID

In the remote schema, each remote database represents one store. The synchronization scripts you wrote include logic that instructs the MobiLink server to download a subset of data based on the remote ID of the remote database. You must set the database's remote ID to the value of a valid store identifier.

It is important to complete this step before the first synchronization because when the remote device synchronizes for the first time, it downloads all information related to the store (in this case, Thoreau Reading Discount Chain).

To set the remote ID to a valid store identifier

1. Choose a valid store identifier.

   a. Connect to the pubs2 database as **sa**, using the isql utility in Adaptive Server Enterprise. At a command prompt, run the following command, all on one line:

      ```
      isql
      -U sa
      -P Your password for sa account
      -D pubs2
      ```

      If you are accessing Adaptive Server Enterprise remotely, use the -S parameter to specify the server name.

   b. To view a list of valid store identifiers in the stores table, execute the following statement:

      ```
      SELECT * FROM stores
      ```

      In this tutorial, the remote database represents the Thoreau Reading Discount Chain store, which has a value of 5023 for its store identifier.

c. To exit Adaptive Server Enterprise, run the following command:

```
exit
```

2. To set the database's remote ID to 5023, run the following command, all on one line:

```
dbisql
    -c "server=remote_eng;dbn=remote_db;uid=DBA;pwd=sql"
    "SET OPTION PUBLIC.ml_remote_id='5023'"
```

- **dbisql** is the application used to execute SQL commands against a SQL Anywhere database.
- **eng** specifies the database server name remote_eng.
- **dbn** specifies the database name remote_db.
- **uid** is the user name used to connect to your remote database.
- **pwd** is the password used to connect to your remote database.
- **SET OPTION PUBLIC.ml_remote_id='5023'** is the SQL command used to set the remote ID to 5023.

Further reading

- "The SQL Anywhere database server" [*SQL Anywhere Server - Database Administration*]
- "Synchronizing a deployed model" on page 41
- "Running the MobiLink server" [*MobiLink - Server Administration*]
- "Remote IDs" [*MobiLink - Client Administration*]

Lesson 7: Synchronize

Now you are ready to synchronize the remote client for the first time. This is done with the MobiLink client program dbmlsync. Dbmlsync connects to the remote database, authenticates itself with the MobiLink server, and performs all the uploads and downloads necessary to synchronize the remote and consolidated databases based on a publication in the remote database.

To synchronize the remote client

- At a command prompt, run the following command, all on one line:

```
dbmlsync -c "server=remote_eng;dbn=remote_db;uid=DBA;pwd=sql;"
    -n sync_ase_publication
    -u ase_remote -mp ase_pass
```

- **dbmlsync** is the synchronization application.
- **server=remote_eng** specifies the name of the remote database server.
- **dbn=remote_db** specifies the name of the remote database.
- **uid** is the user name used to connect to the remote database.
- **pwd** is the password used to connect to the remote database.

- **sync_ase_publication** is the name of the publication on the remote device that is used to perform the synchronization. (This publication was created by the **Create Synchronization Model Wizard**.)
- **ase_remote** is the user name used to authenticate with the MobiLink server.
- **ase_pass** is the password used to authenticate with the MobiLink server.

The progress of the synchronization is displayed in the **SQL Anywhere MobiLink Client Messages** window. When this command runs successfully, the dbmlsync application populates the remote database with a subset of information from the consolidated database.

If synchronization fails, check the connection information you pass to the dbmlsync application, and the MobiLink user name and password. If the problem persists, check the publication name you used, and ensure the consolidated database and MobiLink server are running. You can also examine the contents of the synchronization logs (server and client).

Note
If you are running the dbmlsync application on a different computer from your MobiLink server, you must also pass in arguments that specify the location of the MobiLink server.

View the data

After successfully synchronizing the remote client to the consolidated database through the MobiLink server, the remote data should be populated with information relevant to one store. You can verify this in Sybase Central using the SQL Anywhere 12 plug-in.

To view the data in the remote database

1. Start Sybase Central.
2. Connect to the remote database:
   a. On the left pane, right-click **SQL Anywhere 12** and choose **Connect**.
   b. In the **Authentication** dropdown list, choose **Database**, and then:
      i. In the **User ID** field, type **DBA**.
      ii. In the **Password** field, type **sql**.
   c. From the **Action** dropdown list, choose **Connect To A Running Database On This Computer**.
   d. In the**Server Name** field, type **remote_eng** and in the **Database Name** field, type **remote_db**.
   e. Click **Connect**.
3. If the tables created from the consolidated database are not visible, perform the following steps:
   a. Right-click *remote_db* and **Configure Owner Filter**.
   b. Select **dbo** and click **OK**.

      The tables created from the consolidated database appear in the left pane. Ownership of these tables by dbo is preserved in the remote database.

4. Choose any remote table and then click the **Data** tab on the right pane.

   In the sales, salesdetail and stores tables, all the records are for the store with an identifier of 5023. This particular store is not concerned with the sales information of other stores. For this reason, you set the synchronization scripts to filter out rows by the remote ID, and you set this database's remote ID to the value of a particular store identifier. Now this particular store's database takes up less space, and requires less time to synchronize. Since the remote database size is kept to a minimum, frequently performed operations such as entering a new sale or processing a refund on a previous sale run faster and more efficiently.

Further reading

- "The synchronization process" on page 13
- "dbmlsync syntax" [*MobiLink - Client Administration*]

Cleanup

Regenerate the pubs2 database and remove all tutorial materials from your computer.

To regenerate the pubs2 database

- To run the script that installs the pubs2 database, run the following command:

```
isql
-U sa
-P Your password for sa account
-i %SYBASE%\%SYBASE_ASE%\scripts\instpbs2
```

  If you are accessing Adaptive Server Enterprise remotely, use the -S parameter to specify the server name. You also have to copy the instpbs2 file locally onto your computer. The -i parameter needs to updated so that the new location of the instpbs2 file is specified.

To delete the synchronization model

1. Start Sybase Central.
2. Double-click **MobiLink 12** in the right pane.

   The **sync_ase** model appears.
3. Right click **sync_ase** and choose **Delete**.

To erase the remote database

- To erase the remote database, use the dberase utility. Run the following command:

```
dberase sync_ase\remote\sync_ase_remote.db
```

Tutorial: Using Java synchronization logic

Introduction to Java Synchronization logic

This tutorial guides you through the basic steps for using Java synchronization logic. Using the CustDB sample as a SQL Anywhere consolidated database, you specify simple class methods for MobiLink table-level events. The process also involves running the MobiLink server (mlsrv12) with an option to set the path of compiled Java classes.

Required software

- SQL Anywhere 12
- Java Software Development Kit

Competencies and experience

You should have:

- Familiarity with Java
- Basic knowledge of MobiLink event scripts and MobiLink synchronization

Goals

You gain competence and familiarity with:

- Using Java class methods for MobiLink table-level event scripts

Key concepts

This section uses the following steps to implement Java synchronization using the CustDB sample database:

- Compiling a source file with MobiLink server API references
- Specifying class methods for table-level events
- Running the MobiLink server (mlsrv12) with the -sl java option
- Testing synchronization with a sample Windows client application

Suggested background reading

For more information about synchronization scripts, see "Writing synchronization scripts" [*MobiLink - Server Administration*].

Lesson 1: Compile the CustdbScripts Java class with MobiLink references

Java classes encapsulate synchronization logic in methods.

In this lesson, you compile a class associated with the CustDB sample database.

MobiLink database sample

SQL Anywhere ships with a SQL Anywhere sample database (CustDB) that is already set up for synchronization, including the SQL scripts required for synchronization. The CustDB ULCustomer table, for example, is a synchronized table that supports a variety of table-level events.

CustDB is designed to be a consolidated database server for both UltraLite and SQL Anywhere clients. The CustDB database has a DSN named SQL Anywhere 12 CustDB.

Create a CustdbScripts class

In this section, you create a Java class called CustdbScripts with logic to handle the ULCustomer upload_insert and download_cursor events. You enter the CustdbScripts code in a text editor and save the file as *CustdbScripts.java*.

To create a CustdbScripts class

1. Create a directory for the Java class and assembly.

   This tutorial assumes the path *c:\mljava*.

2. Using a text editor, enter the following code:

   Note
   The sample code in this tutorial illustrates a deprecated coding practice. It is recommended that you do not return SQL code when writing Java or .NET scripts. You should use make changes to the consolidated database directly from Java or .NET.

   For more information, see "Direct row handling" [*MobiLink - Server Administration*].

```
public class CustdbScripts {
    public static String UploadInsert() {
        return("INSERT INTO ULCustomer(cust_id,cust_name) values (?,?)");
    }
    public String DownloadCursor(java.sql.Timestamp ts,String user ) {
        return("SELECT cust_id, cust_name FROM ULCustomer where
last_modified >= ' " + ts + " ' ");
    }
}
```

   Note
   When creating your own custom scripts, make sure that the classes and associated methods are defined as **public**.

3. Save the file as *CustdbScripts.java* in *c:\mljava*.

Compile the Java source code

To execute Java synchronization logic, the MobiLink server must have access to the classes in *mlscript.jar*. This JAR file contains a repository of MobiLink server API classes to use in your Java methods.

When compiling Java source code for MobiLink, you must include *mlscript.jar* to make use of the MobiLink server API. In this section, you use the javac utility's -classpath option to specify *mlscript.jar* for the CustdbScripts class.

To compile the Java source code

1. At a command prompt, navigate to the folder containing *CustdbScripts.java* (*c:\mljava*).
2. Compile the file.

   Run the following command:

   ```
   javac custdbscripts.java -classpath "C:\Program Files\SQL Anywhere 12\java
   \mlscript.jar"
   ```

   Replace C:\Program Files\SQL Anywhere 12 with the location of your SQL Anywhere installation.

The *CustdbScripts.class* file is generated.

Further reading

For more information about the MobiLink server API for Java, see "MobiLink server API for Java reference" [*MobiLink - Server Administration*].

For more information about Java methods, see "Methods" [*MobiLink - Server Administration*].

Lesson 2: Specify class methods to handle events

CustdbScripts.class, created in the previous lesson, encapsulates the methods UploadInsert and DownloadCursor. These methods contain implementations for the ULCustomer upload_insert and download_cursor events, respectively.

In this section, you specify class methods for table-level events using two approaches:

1. Creating a MobiLink project through Sybase Central.

   You connect to the CustDB database using Sybase Central, replace the upload_insert SQL script with a Java script, and then specify CustdbScripts.UploadInsert to handle the event.

2. Using the ml_add_java_table_script stored procedure:

   You connect to the CustDB database with Interactive SQL and run ml_add_java_table_script, specifying CustdbScripts.DownloadCursor to handle the download_cursor event.

To create a new MobiLink project

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.
2. From the **Tools** menu, choose **MobiLink 12** » **New Project**.
3. In the **Name** field, type **mljava_project**.
4. In the **Location** field, type *C:\mljava*, and then click **Next**.
5. Check the **Add A Consolidated Database To The Project** option.

6. In the **Database Display Name** field, type **mljava_db**.
7. Click **Edit**. The **Connect To A Generic ODBC Database** window appears.
8. In the **ODBC Data Source** name field, click **Browse** and select **SQL Anywhere 12 CustDB**.
9. Click **OK**, then click **Save**.
10. Click **Finish**.
11. Click **OK**.

In the next procedure, you use Sybase Central to specify a Java method as the script for the ULCustomer upload_insert event.

To subscribe CustdbScripts.UploadInsert to the upload_insert event for the ULCustomer table

1. From the **View** menu, choose **Folders**.
2. In the left pane of Sybase Central under **MobiLink 12**, expand **mljava_project**, **Consolidated Databases**, **mljava_db**, **Synchronized Tables** and then **ULCustomer**.
3. In the right pane, select the custdb 12.0 upload_insert table script. From the **Edit** menu, choose **Delete**.
4. Create a new upload_insert table script.
   a. From the **File** menu, choose **New** » **Table Script**.
   b. In the **For Which Version Do You Want To Create The Table Script** list, click **custdb 12.0**.
   c. In the **Which Event Should Cause The Table Script To Be Executed** list, click **upload_insert**, and then click **Next**.
   d. Select **Create a new script definition** and then select **Java**.
   e. Click **Finish**.
5. Enter the Java method name to load for the custdb 12.0 upload_insert table script.

   In the right pane of Sybase Central, use the following Java script for the **upload_insert** event:

   ```
   CustdbScripts.UploadInsert
   ```
6. From the **File** menu, choose **Save** to save the script.
7. Close Sybase Central.

Alternatively, you can use the ml_add_java_connection_script and ml_add_java_table_script stored procedures. Using these stored procedures is more efficient, especially if you have a large number of .NET methods to handle synchronization events.

See “ml_add_java_connection_script system procedure” [*MobiLink - Server Administration*] and “ml_add_java_table_script system procedure” [*MobiLink - Server Administration*].

In the next procedure, you connect to CustDB with Interactive SQL and execute ml_add_java_table_script, assigning CustdbScripts.DownloadCursor to the download_cursor event.

To specify CustdbScripts.DownloadCursor() to handle the ULCustomer download_cursor event

1. Connect to the sample database with Interactive SQL.
   a. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Interactive SQL**, or run the following command:

      ```
      dbisql
      ```

   b. Click **ODBC Data Source Name** and type SQL Anywhere 12 CustDB.
   c. Click **Connect**.

2. Run the following SQL script in Interactive SQL:

   ```
   CALL ml_add_java_table_script(
       'custdb 12.0',
       'ULCustomer',
       'download_cursor',
       'CustdbScripts.DownloadCursor');
   COMMIT;
   ```

   The following is a description of each parameter:

Parameter	Description
custdb 12.0	The script version.
ULCustomer	The synchronized table.
download_cursor	The event name.
CustdbScripts.DownloadCursor	The fully qualified Java method.

Note
Your updated download_cursor script might not appear in Sybase Central. To view the most recent changes to the MobiLink project in Sybase Central, click the **View** menu and choose **Refresh All**.

3. Close Interactive SQL.

In this lesson, you specified Java methods to handle ULCustomer table-level events. The next lesson ensures that the MobiLink server loads the appropriate class files and the MobiLink server API.

Further reading

For more information about adding and deleting synchronization scripts, see “Adding and deleting scripts” [*MobiLink - Server Administration*].

For more information about adding scripts with stored procedures, see "ml_add_java_connection_script system procedure" [*MobiLink - Server Administration*] and "ml_add_java_table_script system procedure" [*MobiLink - Server Administration*].

Lesson 3: Run the MobiLink server with -sl java

Running the MobiLink server with the -sl java -cp option specifies a set of directories to search for class files, and forces the Java VM to load on server startup.

To start the MobiLink server (mlsrv12) and load Java assemblies

- Start the MobiLink server with the -sl java option.

  Run the following command:

  ```
  mlsrv12 -c "dsn=SQL Anywhere 12 CustDB" -sl java (-cp c:\mljava)
  ```

  A message indicating that the server is ready to handle requests appears. The Java method is executed when the upload_insert event triggers during synchronization.

Further reading

For more information about starting the MobiLink server for Java, see "-sl java mlsrv12 option" [*MobiLink - Server Administration*].

Lesson 4: Test synchronization

UltraLite comes with a sample Windows client that automatically invokes the dbmlsync utility when the user issues a synchronization.

Start the application

You run the application against the CustDB consolidated database you started in the previous lesson.

To start and synchronize the sample application

1. Start the sample application.

   Click **Start** » **Programs** » **SQL Anywhere 12** » **UltraLite** » **Windows Sample Application**.

2. Enter an employee ID and synchronize.

   Enter a value of **50** for the employee ID and click **OK**.

The application automatically synchronizes and a set of customers, products, and orders are downloaded to the application from the CustDB consolidated database.

Add an order and synchronize

You enter a new customer name and order details. During a subsequent synchronization, this information is uploaded to the CustDB consolidated database and the upload_insert and download_cursor events for the ULCustomer table triggers.

To add an order

1. From the **Order** menu, choose **New**.
2. Enter **Frank Javac** for a customer name.
3. Choose a product, and enter the quantity and discount.
4. Click **OK** to add the new order.

You have now modified the data in your local UltraLite database. This data is not shared with the consolidated database until you synchronize.

To synchronize with the consolidated database and trigger the upload_insert event

- From the **File** menu, choose **Synchronize**.

  A message indicating the Insert successfully uploaded to the consolidated database appears.

Further reading

For more information about the CustDB Windows Application, see "Exploring the CustDB sample for MobiLink" on page 46.

Cleanup

Remove tutorial materials from your computer.

To remove tutorial materials from your computer

1. Delete your Java source files.

   For example, delete the *c:\mljava* directory.

   Caution
   Make sure you only have tutorial related materials in this directory.

2. Close Interactive SQL and UltraLite Windows client application.

   Choose **Exit** from the **File** menu of each application.

3. Close the SQL Anywhere, MobiLink, and synchronization client windows.

   Right-click each task-bar item and choose **Close**.

4. Reset the database for the Windows Sample Application.

   Run the following command from the *samples-dir\UltraLite\CustDB* directory:

   ```
   makedbs
   ```

Further reading

For more information about writing MobiLink synchronization scripts in Java, see "Setting up Java synchronization logic" [*MobiLink - Server Administration*].

For an example illustrating the use of Java synchronization scripts for custom authentication, see "Java synchronization example" [*MobiLink - Server Administration*].

For more information about synchronization scripting, see "Writing synchronization scripts" [*MobiLink - Server Administration*] and "Synchronization events" [*MobiLink - Server Administration*].

For an introduction to other methods of synchronization such as timestamp, see "Synchronization techniques" [*MobiLink - Server Administration*].

Tutorial: Using .NET synchronization logic

Introduction to .NET synchronization logic

This tutorial guides you through the basic steps for using .NET synchronization logic. Using the CustDB sample as a SQL Anywhere consolidated database, you specify simple class methods for MobiLink table-level events. The process also involves running the MobiLink server (mlsrv12) with an option that sets the path of .NET assemblies.

Required software

- SQL Anywhere 12
- Microsoft .NET Framework SDK

Competencies and experience

You should have:

- Familiarity with .NET
- Basic knowledge of MobiLink event scripts

Goals

You gain competence and familiarity with:

- Using .NET class methods for MobiLink table-level event scripts

Key concepts

This section uses the following steps to implement .NET synchronization using the CustDB sample database:

- Compiling the *CustdbScripts.dll* private assembly with MobiLink references
- Specifying class methods for table-level events
- Running the MobiLink server (mlsrv12) with the -sl dnet option
- Testing synchronization with a sample Windows client application

Suggested background reading

For more information about synchronization scripts, see "Writing synchronization scripts" [*MobiLink - Server Administration*].

Lesson 1: Compile the CustdbScripts.dll assembly with MobiLink references

.NET classes encapsulate synchronization logic in methods.

In this lesson, you compile a class associated with the CustDB sample database.

MobiLink database sample

SQL Anywhere ships with a SQL Anywhere sample database (CustDB) that is already set up for synchronization, including the SQL scripts required for synchronization. The CustDB ULCustomer table, for example, is a synchronized table that supports a variety of table-level events.

CustDB is designed to be a consolidated database server for both UltraLite and SQL Anywhere clients. The CustDB database has a DSN named SQL Anywhere 12 CustDB.

The CustdbScripts Assembly

In this section, you create a .NET class called CustdbScripts with logic to handle the ULCustomer upload_insert and download_cursor events.

MobiLink server API

To execute .NET synchronization logic, the MobiLink server must have access to the classes in *iAnywhere.MobiLink.Script.dll*. *iAnywhere.MobiLink.Script.dll* contains a repository of MobiLink server API for .NET classes to use in your .NET methods.

For more information about the MobiLink server API for .NET, see "MobiLink server .NET API reference (.NET 2.0)" [*MobiLink - Server Administration*].

When compiling the CustdbScripts class, you must include this assembly to make use of the API. You can compile your class using Visual Studio or at a command prompt.

- In Visual Studio, create a new Class Library and enter the CustdbScripts code. Link *iAnywhere.MobiLink.Script.dll*, and build the assembly for your class.
- Put the CustdbScripts code in a text file and save the file as *CustdbScripts.cs* (or *CustdbScripts.vb* for Visual Basic). Using a command line compiler, reference *iAnywhere.MobiLink.Script.dll* and build the assembly for your class.

To create the CustdbScripts assembly using Visual Studio

1. Start a new Visual C# or Visual Basic Windows Class Library project.

   Use **CustdbScripts** for the project Name and enter an appropriate path. This tutorial assumes the path *c:\mldnet*.

2. Ensure that the *Class1.cs* file is highlighted in the Solution Explorer.

3. Enter the CustdbScripts code.

   Note
   The sample code in this tutorial illustrates a deprecated coding practice. It is recommended that you do not return SQL code when writing Java or .NET scripts. You should use make changes to the consolidated database directly from Java or .NET.

   For more information, see "Direct row handling" [*MobiLink - Server Administration*].

   For C#, replace the existing code with the following:

```
namespace MLExample {
    class CustdbScripts {
        public static string UploadInsert() {
            return("INSERT INTO ULCustomer(cust_id,cust_name) values
(?,?)");
        }

        public static string DownloadCursor(System.DateTime ts, string
user ) {
            return("SELECT cust_id, cust_name FROM ULCustomer WHERE
last_modified >= '"
                + ts.ToString("yyyy-MM-dd hh:mm:ss.fff") +"'");
        }
    }
}
```

   For Visual Basic, replace the existing code with the following:

```
Namespace MLExample

 Class CustdbScripts

  Public Shared Function UploadInsert() As String
    Return("INSERT INTO ULCustomer(cust_id,cust_name) values (?,?)")
  End Function

  Public Shared Function DownloadCursor(ByVal ts As System.DateTime, ByVal
user As String) As String
   Return("SELECT cust_id, cust_name FROM ULCustomer " + _
```

```
    "WHERE last_modified >= '" + ts.ToString("yyyy-MM-dd hh:mm:ss.fff")
+"'")
  End Function

 End Class

End Namespace
```

4. Add a reference to the MobiLink server API.
   - From the Visual Studio **Project** menu, choose **Add Existing Item**.
   - Select *iAnywhere.MobiLink.Script.dll* in *C:\Program Files\SQL Anywhere 12\Assembly\V2*. Replace C:\Program Files\SQL Anywhere 12 with the directory of your SQL Anywhere installation.
   - From the **Add** dropdown, choose to add the file as a link.
5. For Visual Basic, right-click the CustdbScripts project and select the table **Properties** » **General**. Make sure that the text field **Root Namespace** is cleared of all text.
6. Build *CustdbScripts.dll*.

   From the **Build** menu, choose **Build CustdbScripts**.

   This creates *CustdbScripts.dll* in *C:\mldnet\CustdbScripts\CustdbScripts\bin\Debug*.

To create the CustdbScripts assembly at a command prompt

1. Create a directory for the .NET class and assembly.

   This tutorial assumes the path *c:\mldnet*.

2. Using a text editor, enter the CustdbScripts code.

> **Note**
> The sample code in this tutorial illustrates a deprecated coding practice. It is recommended that you do not return SQL code when writing Java or .NET scripts. You should use make changes to the consolidated database directly from Java or .NET.
>
> For more information, see "Direct row handling" [*MobiLink - Server Administration*].

For C#, type:

```
namespace MLExample
{
class CustdbScripts
{
  public static string UploadInsert()
 {
   return("INSERT INTO ulcustomer(cust_id,cust_name) values (?,?)");
 }
 public static string DownloadCursor(System.DateTime ts, string user )
 {
   return("SELECT cust_id, cust_name FROM ULCustomer where last_modified
>= '" + ts.ToString("yyyy-MM-dd hh:mm:ss.fff") +"'");
 }
```



```
}
}
```

For Visual Basic, type:

```
Namespace MLExample

 Class CustdbScripts

  Public Shared Function UploadInsert() As String
    Return("INSERT INTO ULCustomer(cust_id,cust_name) values (?,?)")
  End Function

  Public Shared Function DownloadCursor(ByVal ts As System.DateTime, ByVal
user As String) As String
   Return("SELECT cust_id, cust_name FROM ULCustomer " + _
    "WHERE last_modified >= '" + ts.ToString("yyyy-MM-dd hh:mm:ss.fff")
+"'")
  End Function

 End Class

End Namespace
```

3. Save the file as *CustdbScripts.cs* (*CustdbScripts.vb* for Visual Basic) in *c:\mldnet*.

4. Compile the file.

   For C#, run the following command:

   ```
   csc /out:c:\mldnet\custdbscripts.dll /target:library /reference:"C:
   \Program Files\SQL Anywhere 12\Assembly\v2\iAnywhere.MobiLink.Script.dll"
   c:\mldnet\CustdbScripts.cs
   ```

   For Visual Basic, run the following command:

   ```
   vbc /out:c:\mldnet\custdbscripts.dll /target:library /reference:"C:
   \Program Files\SQL Anywhere 12\Assembly\v2\iAnywhere.MobiLink.Script.dll"
   c:\mldnet\CustdbScripts.vb
   ```

   The *CustdbScripts.dll* assembly is generated.

Further reading

For more information about the MobiLink server API for .NET, see "MobiLink server .NET API reference (.NET 2.0)" [*MobiLink - Server Administration*].

For more information about .NET methods, see "Methods" [*MobiLink - Server Administration*].

Lesson 2: Specify class methods for events

CustdbScripts.dll, created in the previous lesson, encapsulates the methods UploadInsert() and DownloadCursor(). These methods contain implementation for the ULCustomer upload_insert and download_cursor events, respectively.

In this section, you specify class methods for table-level events using two approaches:

1. Creating a MobiLink project through Sybase Central.

   You connect to the CustDB database using Sybase Central, replace the upload_insert SQL script with a .NET script, and then specify MLExample.CustdbScripts.UploadInsert to handle the event.

2. Using the ml_add_dnet_table_script stored procedure.

   You connect to the CustDB database with Interactive SQL and run ml_add_dnet_table_script, specifying MLExample.CustdbScripts.DownloadCursor for the download_cursor event.

To create a new MobiLink project

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.
2. From the **Tools** menu, choose **MobiLink 12** » **New Project**.
3. In the **Name** field, type **mldnet_project**.
4. In the **Location** field, type *C:\mldnet*, and then click **Next**.
5. Check the **Add A Consolidated Database To The Project** option.
6. In the **Database Display Name** field, type **mldnet_db**.
7. Click **Edit**. The **Connect To A Generic ODBC Database** window appears.
8. In the **ODBC Data Source** name field, click **Browse** and select **SQL Anywhere 12 CustDB**.
9. Click **OK**, then click **Save**.
10. Click **Finish**.
11. Click **OK**.

In the next procedure, you use Sybase Central to specify a .NET method as the script for the ULCustomer upload_insert event.

To subscribe CustdbScripts.uploadInsert() to the upload_insert event for the ULCustomer table

1. From the **View** menu, choose **Folders**.
2. In the left pane of Sybase Central under **MobiLink 12**, expand **mldnet_project**, **Consolidated Databases**, **mldnet_db**, **Synchronized Tables** and then **ULCustomer**.
3. In the right pane, select the custdb 12.0 upload_insert table script. From the **Edit** menu, choose **Delete**.
4. Create a new upload_insert table script.
   a. From the **File** menu, choose **New** » **Table Script**.
   b. In the **For Which Version Do You Want To Create The Table Script** list, click **custdb 12.0**.

c. In the **Which Event Should Cause The Table Script To Be Executed** list, click **upload_insert**, and then click **Next**.

d. Select **Create a new script definition** and then select **.NET**.

e. Click **Finish**.

5. Enter the .NET method name to load for the custdb 12.0 upload_insert table script.

   In the right pane of Sybase Central, use the following .NET script for the **upload_insert** event:

   ```
   MLExample.CustdbScripts.UploadInsert
   ```

6. From the **File** menu, choose **Save** to save the script.

7. Close Sybase Central.

Alternatively, you can use the ml_add_dnet_connection_script and ml_add_dnet_table_script stored procedures. Using these stored procedures is more efficient, especially if you have a large number of .NET methods to handle synchronization events.

See "ml_add_dnet_connection_script system procedure" [*MobiLink - Server Administration*] and "ml_add_dnet_table_script system procedure" [*MobiLink - Server Administration*].

In the next procedure, you connect to CustDB with Interactive SQL and execute ml_add_dnet_table_script, assigning MLExample.CustdbScripts.DownloadCursor to the download_cursor event.

To specify MLExample.CustdbScripts.DownloadCursor for the ULCustomer download_cursor event

1. Connect to the sample database with Interactive SQL.

   a. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Interactive SQL**, or run the following command:

      ```
      dbisql
      ```

   b. Click **ODBC Data Source Name** and type SQL Anywhere 12 CustDB.

   c. Click **Connect**.

2. Run the following SQL script in Interactive SQL:

   ```
   CALL ml_add_dnet_table_script(
       'custdb 12.0',
       'ULCustomer',
       'download_cursor',
       'MLExample.CustdbScripts.DownloadCursor');
   COMMIT;
   ```

   The following is a description of each parameter:

Parameter	Description
custdb 12.0	The script version.
ULCustomer	The synchronized table.
download_cursor	The event name.
MLExample.CustdbScripts.DownloadCursor	The fully qualified .NET method.

Note
Your updated download_cursor script might not appear in Sybase Central. To view the most recent changes to the MobiLink project in Sybase Central, click the **View** menu and choose **Refresh All**.

3. Close Interactive SQL.

In this lesson, you specified .NET methods to handle ULCustomer table-level events. The next lesson ensures that the MobiLink server loads the appropriate class files and the MobiLink server API.

Further reading

For more information about adding and deleting synchronization scripts, see "Adding and deleting scripts" [*MobiLink - Server Administration*].

For more information about adding scripts with stored procedures, see "ml_add_dnet_connection_script system procedure" [*MobiLink - Server Administration*] and "ml_add_dnet_table_script system procedure" [*MobiLink - Server Administration*].

Lesson 3: Run the MobiLink server with -sl dnet

Running the MobiLink server with the -sl dnet option specifies the location of .NET assemblies and forces the CLR to load on server startup.

If you compiled using Visual Studio, the location of *CustdbScripts.dll* is *c:\mldnet\CustdbScripts\CustdbScripts\bin\Debug*. If you compiled at a command prompt, the location of *CustdbScripts.dll* is *c:\mldnet*.

To start the MobiLink server (mlsrv12) and load .NET assemblies

- Start the MobiLink server with the -sl dnet option.

  Run the following command if you used Visual Studio to compile your assembly:

  ```
  mlsrv12 -c "dsn=SQL Anywhere 12 CustDB" -dl -o c:\mldnet\cons1.txt -v+ -sl
  dnet(-MLAutoLoadPath=c:\mldnet\CustdbScripts\CustdbScripts\bin\Debug)
  ```

  Run the following command if you compiled your assembly at a command prompt:

  ```
  mlsrv12 -c "dsn=SQL Anywhere 12 CustDB" -dl -o c:\mldnet\cons1.txt -v+ -sl
  dnet(-MLAutoLoadPath=c:\mldnet)
  ```


A message indicating that the server is ready to handle requests appears. The .NET method is now executed when the upload_insert event triggers during synchronization.

Further reading

For more information about starting the MobiLink server for .NET, see "-sl dnet mlsrv12 option" [*MobiLink - Server Administration*].

Lesson 4: Test synchronization

UltraLite comes with a sample Windows client that automatically invokes the dbmlsync utility when the user issues a synchronization.

Start the application

You run the application against the CustDB consolidated database you started in the previous lesson.

To start the sample application and test authentication

1. Start the sample application.

   From the **Start** menu, choose **Programs** » **SQL Anywhere 12** » **UltraLite** » **Windows Sample Application**.

2. Enter an employee ID and synchronize.

   Enter a value of **50** for the employee ID and click **OK**.

The application automatically synchronizes, and a set of customers, products, and orders are downloaded to the application from the CustDB consolidated database.

Add an order and synchronize

You enter a new customer name and order details. During a subsequent synchronization, this information is uploaded to the CustDB consolidated database and the upload_insert and download_cursor events for the ULCustomer table triggers.

To add an order

1. From the **Order** menu, choose **New**.
2. Enter **Frank DotNET** for a customer name.
3. Choose a product, and enter the quantity and discount.
4. Click **OK** to add the new order.

You have now modified the data in your local UltraLite database. This data is not shared with the consolidated database until you synchronize.

To synchronize with the consolidated database and trigger the upload_insert event

- From the **File** menu, choose **Synchronize**.

  A message indicating the Insert successfully uploaded to the consolidated database appears.

Further reading

For more information about the CustDB Windows Application, see "Exploring the CustDB sample for MobiLink" on page 46.

Cleanup

Remove tutorial materials from your computer.

To remove tutorial materials from your computer

1. Delete your .NET source files.

   For example, delete the *c:\mldnet* directory.

   Caution
   Make sure you only have tutorial related materials in this directory.

2. Close Interactive SQL and UltraLite Windows client application.

   Choose **Exit** from the **File** menu of each application.

3. Close the SQL Anywhere, MobiLink, and synchronization client windows.

   Right-click each task-bar item and choose **Close**.

4. Reset the database for the Windows Sample Application.

   Run the following command from the *samples-dir\UltraLite\CustDB* directory:

   ```
   makedbs
   ```

Further reading

For more information about writing MobiLink synchronization scripts in .NET, see "Setting up .NET synchronization logic" [*MobiLink - Server Administration*].

For information about debugging .NET synchronization logic, see "Debugging .NET synchronization logic" [*MobiLink - Server Administration*].

For a detailed example illustrating the use of .NET synchronization scripts for custom authentication, see ".NET synchronization example" [*MobiLink - Server Administration*].

For more information about synchronization scripting, see "Writing synchronization scripts" [*MobiLink - Server Administration*] and "Synchronization events" [*MobiLink - Server Administration*].

For an introduction to other methods of synchronization such as timestamp, see "Synchronization techniques" [*MobiLink - Server Administration*].

Tutorial: Using Java or .NET for custom user authentication

Introduction to MobiLink custom authentication

MobiLink synchronization scripts can be written in SQL, Java, or .NET. You can use Java or .NET to add custom actions at any point of a synchronization.

In this tutorial, you add a Java or .NET method for the authenticate_user connection event. The authenticate_user event allows you to specify a custom authentication scheme and override the MobiLink built-in client authentication.

Required software

- SQL Anywhere 12
- Java Software Development Kit or the Microsoft .NET Framework

Competencies and experience

You require:

- Familiarity with Java or .NET
- Basic knowledge of MobiLink event scripts and MobiLink synchronization

Goals

You gain competence and familiarity with:

- MobiLink custom authentication

Key concepts

This section uses the following steps to implement basic Java-based synchronization using the MobiLink CustDB sample database:

- Compiling a source file with MobiLink server API references
- Specifying class methods for particular table-level events
- Running the MobiLink server (mlsrv12) with the -sl option

- Testing synchronization with a sample Windows client application

Suggested background reading

For more information about authenticating MobiLink clients, see "Choosing a user authentication mechanism" [*MobiLink - Client Administration*].

For more information about integrating POP3, IMAP, or LDAP authentication, see "Authenticating to external servers" [*MobiLink - Client Administration*].

For more information about .NET or Java synchronization scripts, see "Writing synchronization scripts in .NET" [*MobiLink - Server Administration*] or "Writing synchronization scripts in Java" [*MobiLink - Server Administration*].

Lesson 1: Create a Java or .NET class for custom authentication (server-side)

In this lesson, you compile a class containing Java or .NET logic for custom authentication.

Custom authentication using the MobiLink server Java API

The MobiLink server must have access to the classes in *mlscript.jar* to execute Java synchronization logic. *mlscript.jar* contains a repository of MobiLink Java server API classes to utilize in your Java methods. When you compile your Java class, you reference *mlscript.jar*.

To create a Java class for custom authentication

1. Create a class named MobiLinkAuth and write an authenticateUser method.

   The MobiLinkAuth class includes the authenticateUser method used for the authenticate_user synchronization event. The authenticate_user event provides parameters for the user and password. You return the authentication result in the authentication_status inout parameter.

   Note
   You register the authenticateUser method for the authenticate_user synchronization event in "Lesson 2: Register your Java or .NET scripts for the authenticate_user event" on page 144.

   Use the following code for your server application:

```
import ianywhere.ml.script.*;

public class MobiLinkAuth {
    public void authenticateUser (
        ianywhere.ml.script.InOutInteger authentication_status,
        String user,
        String pwd,
        String newPwd ) {

        if (user.substring(0,2).equals("128")) {
            // success: an auth status code of 1000
            authentication_status.setValue(1000);
        } else {
```



```
                // fail: an authentication_status code of 4000
                authentication_status.setValue(4000);
            }
        }
    }
```

This code illustrates a simple case of custom user authentication. Authentication succeeds when the client accesses the consolidated database using a user name that starts with "128".

2. Save your code.

   This tutorial assumes *c:\MLauth* as the working directory for server-side components. Save the file as *MobiLinkAuth.java* in this directory.

3. Compile your class file.

   a. Navigate to the directory that contains your Java file.

   b. Compile the MobiLinkAuth and refer to the MobiLink server Java API library

      Run the following command, replacing *C:\Program Files\SQL Anywhere 12* with your SQL Anywhere 12 directory:

      ```
      javac MobiLinkAuth.java -classpath "C:\Program Files\SQL Anywhere
      12\java\mlscript.jar"
      ```

      The *MobiLinkAuth.class* file is generated.

Custom authentication using the MobiLink server .NET API

The MobiLink server must have access to the classes in *iAnywhere.MobiLink.Script.dll* to execute .NET synchronization logic. *iAnywhere.MobiLink.Script.dll* contains a repository of MobiLink .NET server API classes to utilize in your .NET methods. When you compile your .NET class, you reference *iAnywhere.MobiLink.Script.dll*.

To create a .NET class for custom authentication

1. Create a class named MobiLinkAuth and write an authenticateUser method.

   The MobiLinkAuth class includes the authenticateUser method used for the authenticate_user synchronization event. The authenticate_user event provides parameters for the user and password. You return the authentication result in the authentication_status inout parameter.

   Note
   You register the authenticateUser method for the authenticate_user synchronization event in “Lesson 2: Register your Java or .NET scripts for the authenticate_user event” on page 144.

   Use the following code for your server application:

   ```
   using iAnywhere.MobiLink.Script;

   public class MobiLinkAuth {
       public void authenticateUser(
           ref int authentication_status,
           string user,
           string pwd,
   ```

```
            string newPwd ) {

            if(user.Substring(0,2)=="128") {
                // success: an auth status code of 1000
                authentication_status = 1000;
            } else {
                // fail: and authentication_status code of 4000
                authentication_status = 4000;
            }
        }
    }
```

This code illustrates a simple case of custom user authentication. Authentication succeeds when the client accesses the consolidated database using a user name that starts with "128".

2. Save your code.

   This tutorial assumes *c:\MLauth* as the working directory for server-side components. Save the file as *MobiLinkAuth.cs* in this directory.

3. Compile your class file.

   a. Navigate to the directory that contains your C# file.

   b. Compile the MobiLinkAuth and refer to the MobiLink server .NET API library

      Run the following command, replacing *C:\Program Files\SQL Anywhere 12* with your SQL Anywhere 12 directory:

      ```
      csc /out:MobiLinkAuth.dll /target:library /reference:"C:\Program Files
      \SQL Anywhere 12\Assembly\v2\iAnywhere.MobiLink.Script.dll"
      MobiLinkAuth.cs
      ```

      The *MobiLinkAuth.dll* assembly is generated.

Further reading

For more information about the authenticate_user event, including a table of authentication_status return codes, see "authenticate_user connection event" [*MobiLink - Server Administration*].

For more information about implementing custom authentication using Java or .NET, see "Java and .NET user authentication" [*MobiLink - Client Administration*].

For a detailed example of Java custom authentication, see "Java synchronization example" [*MobiLink - Server Administration*].

For a detailed example of .NET custom authentication, see ".NET synchronization example" [*MobiLink - Server Administration*].

Lesson 2: Register your Java or .NET scripts for the authenticate_user event

In this lesson, you register the MobiLinkAuth authenticateUser method for the authenticate_user synchronization event. You add this script to CustDB, the MobiLink sample database.

MobiLink database sample

SQL Anywhere ships with a SQL Anywhere sample database (CustDB) that is already set up for synchronization. The CustDB ULCustomer table, for example, is a synchronized table supporting a variety of table-level scripts.

CustDB is designed to be a consolidated database server for both UltraLite and SQL Anywhere clients. The CustDB database has a DSN named SQL Anywhere 12 CustDB.

To register the authenticateUser method for the authenticate_user event

1. Connect to the sample database using Interactive SQL.

   Run the following command:

   ```
   dbisql -c "dsn=SQL Anywhere 12 CustDB"
   ```

2. Use the ml_add_java_connection_script or ml_add_dnet_connection_script stored procedure to register the authenticateUser method for the authenticate_user event.

   For Java, run the following SQL script:

   ```
   CALL ml_add_java_connection_script(
       'custdb 12.0',
       'authenticate_user',
       'MobiLinkAuth.authenticateUser');

   COMMIT;
   ```

   For .NET, run the following SQL script:

   ```
   CALL ml_add_dnet_connection_script(
       'custdb 12.0',
       'authenticate_user',
       'MobiLinkAuth.authenticateUser');

   COMMIT;
   ```

In the next lesson, you start the MobiLink server and load your class file or assembly.

Further reading

For general information about adding and deleting synchronization scripts, see “Adding and deleting scripts” [*MobiLink - Server Administration*].

For more information about the ml_add_java_connection_script, see “ml_add_java_connection_script system procedure” [*MobiLink - Server Administration*].

For more information about the ml_add_dnet_connection_script, see “ml_add_dnet_connection_script system procedure” [*MobiLink - Server Administration*].

Lesson 3: Start the MobiLink server

Starting the MobiLink server with the -sl option allows you to specify a set of directories to search for compiled files.

To start the MobiLink server (mlsrv12)

- Connect to the MobiLink sample database and load your Java class or .NET assembly on the mlsrv12 command line.

  Replace *c:\MLauth* with the location of your source files.

  For Java, run the following command:

  ```
  mlsrv12 -c "dsn=SQL Anywhere 12 CustDB" -o serverOut.txt -v+ -sl java(-cp
  c:\MLauth)
  ```

  For .NET, run the following command:

  ```
  mlsrv12 -c "dsn=SQL Anywhere 12 CustDB" -o serverOut.txt -v+ -sl dnet(-
  MLAutoLoadPath=c:\MLauth)
  ```

  The MobiLinkAuth method is executed when the authenticate_user synchronization event occurs.

Further reading

For more information about starting the MobiLink server for Java, see "-sl java mlsrv12 option" [*MobiLink - Server Administration*].

For more information about starting the MobiLink server for .NET, see "-sl dnet mlsrv12 option" [*MobiLink - Server Administration*].

Lesson 4: Test the authentication

UltraLite comes with a sample Windows client that automatically invokes the dbmlsync utility when the user issues a synchronization. You run the application against the CustDB consolidated database you started in the previous lesson.

To start the sample application and test authentication

1. Start the sample application.

   From the **Start** menu, choose **Programs** » **SQL Anywhere 12** » **UltraLite** » **Windows Sample Application**.

2. Enter an invalid employee ID and synchronize.

   In this application, the employee ID is also the MobiLink user name. If the user name does not begin with "128", your logic causes synchronization to fail. Enter a value of **50** for the employee ID and click **OK**.

   An error stating that the authenticate_user script returned 4000 appears in the **MobiLink Server** window.

   A SQLCODE -103 synchronization error indicating an invalid user ID or password appears in the **UltraLite CustDB Demo** window.

Further reading

For more information about the CustDB Windows Application, see "Exploring the CustDB sample for MobiLink" on page 46.

Cleanup

Remove tutorial materials from your computer.

To remove tutorial materials from your computer

1. Delete your Java or .NET source files.

   For example, delete the *c:\mlauth* directory.

   Caution
   Make sure you only have tutorial related materials in this directory.

2. Close Interactive SQL and UltraLite Windows client application.

   Choose **Exit** from the **File** menu of each application.

3. Close the SQL Anywhere, MobiLink, and synchronization client windows.

   Right-click each task-bar item and choose **Close**.

4. Reset the database for the Windows Sample Application.

   Run the following command from the *samples-dir\UltraLite\CustDB* directory:

   ```
   makedbs
   ```

Further reading

For more information about Java synchronization logic, see "Writing synchronization scripts in Java" [*MobiLink - Server Administration*].

For more information about .NET synchronization logic, see "Writing synchronization scripts in .NET" [*MobiLink - Server Administration*].

For a detailed example illustrating the use of Java or .NET synchronization scripts for custom authentication, see "Java synchronization example" [*MobiLink - Server Administration*] or ".NET synchronization example" [*MobiLink - Server Administration*], respectively.

For information about debugging Java or .NET synchronization logic, see "Debugging Java classes" [*MobiLink - Server Administration*] or "Debugging .NET synchronization logic" [*MobiLink - Server Administration*], respectively.

For more information about synchronization scripts, see "Writing synchronization scripts" [*MobiLink - Server Administration*] and "Synchronization events" [*MobiLink - Server Administration*].

Tutorial: Introduction to direct row handling

Introduction to direct row handling

This tutorial shows you how to implement MobiLink direct row handling so that you can use a data source other than a supported consolidated database.

You can use direct row handling to communicate remote data to any central data source, application, or web service.

This tutorial shows you how to use the MobiLink server APIs for Java and .NET for simple direct row handling. You synchronize the client RemoteOrders table with the consolidated database and add special direct row handling processing for the OrderComments table.

You set up a simple synchronization for the RemoteOrders table.

Required software

- SQL Anywhere 12
- Java Software Development Kit or the Microsoft .NET Framework

Competencies and experience

You require:

- Familiarity with Java or .NET
- Basic knowledge of MobiLink event scripts and MobiLink synchronization

Goals

You gain competence and familiarity with:

- The MobiLink server APIs for Java or .NET
- Creating methods for MobiLink direct row handling

Suggested background reading

- "Understanding MobiLink synchronization" on page 1
- "Synchronization techniques" [*MobiLink - Server Administration*]
- "Direct row handling" [*MobiLink - Server Administration*]

MobiLink code exchange samples are located at http://www.sybase.com/detail?id=1058600#319.

You can post questions on the MobiLink newsgroup: sybase.public.sqlanywhere.mobilink.

Lesson 1: Set up a text file datasource

In this section, you create a new text file to store order information.

To set up a text file datasource

1. Create a new blank text file.
2. Save the file in your working directory.

   This tutorial assumes *c:\MLdirect* as the working directory for server-side components. Save the file as *orderResponses.txt* in this directory.

This file contains responses to comments. For example, *orderResponses.txt* can contain the following entries including tab-delimited values representing the comment_id, order_id, and order_comment:

```
...
786  34  OK, ship promotional material.
787  35  Yes, the product is going out of production.
788  36  No, your commission can not be increased...
...
```

Lesson 2: Set up your MobiLink consolidated database

Your MobiLink consolidated database is a central repository of data and includes MobiLink system tables and stored procedures used to manage the synchronization process. With direct row handling, you synchronize with a data source other than a consolidated database, but you still need a consolidated database to maintain information used by the MobiLink server.

In this lesson, you:

- Create a database and define an ODBC data source.
- Add data tables to synchronize to remote clients.
- Install MobiLink system tables and stored procedures.

> **Note**
> If you already have a MobiLink consolidated database set up with MobiLink system objects and a DSN, you can skip this lesson.

Create your consolidated database

In this tutorial, you create a SQL Anywhere database using the Sybase Central **Create Database Wizard**.

To create your SQL Anywhere RDBMS

1. Click **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.
2. Click **Tools** » **SQL Anywhere 12** » **Create Database**.

3. Click **Next**.
4. Leave the default of **Create Database On This Computer**, and then click **Next**.
5. In the **Save The Main Database File To The Following File** field, type the file name and path for the database. For example, *c:\MLdirect\MLconsolidated.db*.
6. Follow the remaining instructions in the **Create Database Wizard** and accept the default values. On the **Connect To The Database** page, clear the **Stop The Database After Last Disconnect** option.
7. Click **Finish**.

   The MLconsolidated database appears in Sybase Central.
8. Click **Close** on the **Creating Database** window.

Define an ODBC data source for the consolidated database

Use the SQL Anywhere 12 driver to define an ODBC data source for the MLconsolidated database.

To define an ODBC data source for the consolidated database

1. From the Sybase Central **Tools** menu, choose **SQL Anywhere 12** » **Open ODBC Administrator**.
2. Click the **User DSN** tab, and click **Add**.
3. In the **Create New Data Source** window, click **SQL Anywhere 12** and click **Finish**.
4. Perform the following tasks in the **ODBC Configuration For SQL Anywhere** window:
   a. Click the **ODBC** tab.
   b. In the **Data Source Name** field, type **mldirect_db**.
   c. Click the **Login** tab.
   d. In the **User ID** field, type **DBA**.
   e. In the **Password** field, type **sql**.
   f. In the **Server Name** field, type **MLconsolidated**.
   g. Click **OK**.
5. Close ODBC data source administrator.

   Click **OK** on the **ODBC Data Source Administrator** window.

Create tables for synchronization

In this procedure, you create the RemoteOrders table in the MobiLink consolidated database. The RemoteOrders table contains the following columns:

Column	Description
order_id	A unique identifier for orders.
product_id	A unique identifier for products.
quantity	The number of items sold.
order_status	The order status.
last_modified	The last modification date of a row. You use this column for timestamp based downloads, a common technique used to filter rows for efficient synchronization.

To create the RemoteOrders table

1. Connect to your database using Interactive SQL.

   You can start Interactive SQL from Sybase Central or from a command prompt.

   - To start Interactive SQL from Sybase Central, right-click the **MLconsolidated - DBA** database and click **Open Interactive SQL**.
   - To start Interactive SQL at a command prompt, run the following command:

     ```
     dbisql -c "dsn=mldirect_db"
     ```

2. Run the following SQL script in Interactive SQL to create the RemoteOrders table.

   ```
   CREATE TABLE RemoteOrders (
       order_id           INTEGER NOT NULL,
       product_id         INTEGER NOT NULL,
       quantity           INTEGER,
       order_status       VARCHAR(10) DEFAULT 'new',
       last_modified      TIMESTAMP DEFAULT CURRENT TIMESTAMP,
       PRIMARY KEY(order_id)
   )
   ```

   Interactive SQL creates the RemoteOrders table in your consolidated database.

Stay connected in Interactive SQL for the next procedure.

Run MobiLink setup scripts

You can find setup scripts for each supported consolidated database in the *MobiLink/setup* subdirectory of your SQL Anywhere 12 installation.

In this procedure you set up a SQL Anywhere consolidated database. You do this using the *syncsa.sql* setup script. Running *syncsa.sql* creates a series of system tables and stored procedures prefaced with **ml_**. The MobiLink server works with these tables and stored procedures in the synchronization process.

To install MobiLink system tables

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

```
dbisql -c "dsn=mldirect_db"
```

2. Run the following command in Interactive SQL to create MobiLink system tables and stored procedures. Replace *C:\Program Files\SQL Anywhere 12* with the location of your SQL Anywhere 12 installation.

   ```
   READ "C:\Program Files\SQL Anywhere 12\MobiLink\setup\syncsa.sql"
   ```

   Interactive SQL applies *syncsa.sql* to your consolidated database.

Stay connected in Interactive SQL for the next lesson.

Further reading

For information about creating SQL Anywhere databases, see "Initialization utility (dbinit)" [*SQL Anywhere Server - Database Administration*].

For information about creating tables, see "CREATE TABLE statement" [*SQL Anywhere Server - SQL Reference*].

For information about setting up MobiLink consolidated databases, see "MobiLink consolidated databases" [*MobiLink - Server Administration*].

Lesson 3: Add synchronization scripts

In this lesson you add scripts to your consolidated database for SQL row handling and direct row handling.

SQL row handling

SQL row handling allows you to synchronize remote data with tables in your MobiLink consolidated database. SQL-based scripts define:

- How data that is uploaded from a MobiLink client is to be applied to the consolidated database.
- What data should be downloaded from the consolidated database.

In this lesson, you write synchronization scripts for the following SQL-based upload and download events.

- **upload_insert** This event defines how new orders inserted in a client database should be applied to the consolidated database.
- **download_cursor** This event defines the orders that should be downloaded to remote clients.
- **download_delete_cursor** This event is required when using synchronization scripts that are not upload-only. You set the MobiLink server to ignore this event for the purpose of this tutorial.

In this procedure, you add synchronization script information to your MobiLink consolidated database using stored procedures.

To add SQL-based scripts to MobiLink system tables

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "dsn=mldirect_db"
   ```

2. Use the ml_add_table_script stored procedure to add SQL-based table scripts for the upload_insert, download_cursor and download_delete_cursor events.

   Run the following SQL script in Interactive SQL. The upload_insert script inserts the uploaded order_id, product_id, quantity, and order_status into the MobiLink consolidated database. The download_cursor script uses timestamp-based filtering to download updated rows to remote clients.

   ```
   CALL ml_add_table_script( 'default', 'RemoteOrders',
       'upload_insert',
       'INSERT INTO RemoteOrders( order_id, product_id, quantity,
   order_status)
        VALUES( ?, ?, ?, ? )' );

   CALL ml_add_table_script( 'default', 'RemoteOrders',
       'download_cursor',
       'SELECT order_id, product_id, quantity, order_status
        FROM RemoteOrders WHERE last_modified >= ?');

   CALL ml_add_table_script( 'default', 'RemoteOrders',
       'download_delete_cursor', '--{ml_ignore}');

   COMMIT
   ```

Direct row handling processing

In this tutorial, you use direct row handling to add special processing to a SQL-based synchronization system. In this procedure you register method names corresponding to the handle_UploadData, handle_DownloadData, end_download, download_cursor, and download_delete_cursor events. You create your own Java or .NET class in “Lesson 4: Create a Java or .NET class for MobiLink direct row handling” on page 155.

To add information for direct row handling in MobiLink system tables

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "dsn=mldirect_db"
   ```

2. Register a Java or .NET method for the end_download event.

   You use this method to free memory resources when the MobiLink server runs the end_download connection event.

   For Java, run the following SQL script in Interactive SQL:

   ```
   CALL ml_add_java_connection_script( 'default',
       'end_download',
       'MobiLinkOrders.EndDownload' );
   ```

For .NET, run the following SQL script in Interactive SQL:

```
CALL ml_add_dnet_connection_script( 'default',
    'end_download',
    'MobiLinkOrders.EndDownload' );
```

Interactive SQL registers the user-defined EndDownload method for the end_download event.

3. Register Java or .NET methods for the handle_UploadData and handle_DownloadData events.

   For Java, run the following SQL script in Interactive SQL:

   ```
   CALL ml_add_java_connection_script( 'default',
       'handle_UploadData',
       'MobiLinkOrders.GetUpload' );

   CALL ml_add_java_connection_script( 'default',
       'handle_DownloadData',
       'MobiLinkOrders.SetDownload' );
   ```

   For .NET, run the following SQL script in Interactive SQL:

   ```
   CALL ml_add_dnet_connection_script( 'default',
       'handle_UploadData',
       'MobiLinkOrders.GetUpload' );

   CALL ml_add_dnet_connection_script( 'default',
       'handle_DownloadData',
       'MobiLinkOrders.SetDownload' );
   ```

   Interactive SQL registers the user-defined GetUpload and SetDownload methods for the handle_UploadData and handle_DownloadData events, respectively. You create these methods in an upcoming lesson.

4. Register download_cursor and download_delete_cursor events.

   Run the following SQL script in Interactive SQL:

   ```
   CALL ml_add_table_script( 'default', 'OrderComments',
       'download_cursor', '--{ml_ignore}');

   CALL ml_add_table_script( 'default', 'OrderComments',
       'download_delete_cursor', '--{ml_ignore}');
   ```

   The download_cursor and download_delete_cursor events must be registered for the OrderComments table when using scripts because the synchronization is bi-directional and not upload-only. See "Required scripts" [*MobiLink - Server Administration*].

5. Commit your changes.

   Run the following SQL script in Interactive SQL:

   ```
   COMMIT;
   ```

You can now close Interactive SQL.

Further reading

For information about using SQL-based events to upload data from remote clients to a MobiLink consolidated database, see:

- "Writing scripts to upload rows" [*MobiLink - Server Administration*]
- "upload_insert table event" [*MobiLink - Server Administration*]
- "upload_update table event" [*MobiLink - Server Administration*]
- "upload_delete table event" [*MobiLink - Server Administration*]

For information about uploading data to data sources other than consolidated databases, see "Handling direct uploads" [*MobiLink - Server Administration*].

For information about using SQL-based events to download data from a MobiLink consolidated database, see:

- "Writing scripts to download rows" [*MobiLink - Server Administration*]
- "download_cursor table event" [*MobiLink - Server Administration*]
- "download_delete_cursor table event" [*MobiLink - Server Administration*]

For information about downloading data to data sources other than consolidated databases, see "Handling direct downloads" [*MobiLink - Server Administration*].

For information about the synchronization event sequence, see "Overview of MobiLink events" [*MobiLink - Server Administration*].

For information about synchronization techniques for download filtering, see "Timestamp-based downloads" [*MobiLink - Server Administration*] and "Partitioning rows among remote databases" [*MobiLink - Server Administration*].

For information about managing scripts, see "Adding and deleting scripts" [*MobiLink - Server Administration*].

For information about direct row handling, see "Direct row handling" [*MobiLink - Server Administration*].

Lesson 4: Create a Java or .NET class for MobiLink direct row handling

In this lesson, you use direct row handling to process rows in the OrderComments table in your client database. You add the following methods for direct row handling:

- **GetUpload** You use this method for the handle_UploadData event. GetUpload writes uploaded comments to a file called *orderComments.txt*.
- **SetDownload** You use this method for the handle_DownloadData event. SetDownload uses the *orderResponses.txt* file to download responses to remote clients.
- **EndDownload** You use this method for the end_download event. EndDownload frees memory resources.

The following procedure shows you how to create a Java or .NET class including your methods for processing. For a complete listing, see "Complete MobiLinkOrders code listing (Java)" on page 162 or "Complete MobiLinkOrders code listing (.NET)" on page 164.

To create a Java or .NET class for direct row handling

1. Create a class named MobiLinkOrders in Java or .NET.

   For Java, use the following code:

   ```
   import ianywhere.ml.script.*;
   import java.io.*;
   import java.sql.*;

   public class MobiLinkOrders {
   ```

   For .NET, use the following code:

   ```
   using iAnywhere.MobiLink.Script;
   using System.IO;
   using System.Data;
   using System.Text;

   public class MobiLinkOrders {
   ```

2. Declare a class-level DBConnectionContext instance.

   For Java, use the following code:

   ```
   // Class level DBConnectionContext
   DBConnectionContext _cc;
   ```

   For .NET, use the following code:

   ```
   // Class level DBConnectionContext
   private DBConnectionContext _cc = null;
   ```

   The MobiLink server passes a DBConnectionContext instance to your class constructor. DBConnectionContext encapsulates information about the current connection with the MobiLink consolidated database.

3. Declare objects used for file input and output.

   For Java, declare a java.io.FileWriter and java.io.BufferedReader as follows:

   ```
   // Java objects for file i/o
   FileWriter my_writer;
   BufferedReader my_reader;
   ```

   For .NET, declare a StreamWriter and StreamReader as follows:

   ```
   // Instances for file I/O
   private static StreamWriter my_writer = null;
   private static StreamReader my_reader = null;
   ```

4. Create your class constructor.

Your class constructor sets your class-level DBConnectionContext instance.

For Java, use the following code:

```
public MobiLinkOrders( DBConnectionContext cc )
    throws IOException, FileNotFoundException {
    // Declare a class-level DBConnectionContext
    _cc = cc;
}
```

For .NET, use the following code:

```
public MobiLinkOrders(DBConnectionContext cc) {
    _cc = cc;
}
```

5. Write the GetUpload method

   The GetUpload method obtains an UploadedTableData class instance representing the OrderComments table. The OrderComments table contains special comments made by remote sales employees. You create this table in "Lesson 6: Set up your MobiLink client database" on page 167. The UploadedTableData getInserts method returns a result set for new order comments. The writeOrderComment method writes out each row in the result set to a text file.

   For Java, use the following code:

```
    public void writeOrderComment( int _commentID, int _orderID, String
_comments )
        throws IOException
    {
        if (my_writer == null)
            // A FileWriter for writing order comments
            my_writer = new FileWriter( "C:\\MLdirect\
\orderComments.txt",true);

        // Write out the order comments to remoteOrderComments.txt
        my_writer.write(_commentID + "\t" + _orderID + "\t" + _comments);
        my_writer.write( "\n" );
        my_writer.flush();
    }

    //  Method for the handle_UploadData synchronization event
    public void GetUpload( UploadData ut )
        throws SQLException, IOException
    {
        // Get an UploadedTableData for OrderComments
        UploadedTableData orderCommentsTbl =
ut.getUploadedTableByName("OrderComments");

        // Get inserts uploaded by the MobiLink client
        ResultSet insertResultSet = orderCommentsTbl.getInserts();

        while ( insertResultSet.next() )
        {
            // Get order comments
            int _commentID = insertResultSet.getInt("comment_id");
            int _orderID = insertResultSet.getInt("order_id");
            String _specialComments =
insertResultSet.getString("order_comment");
            if (_specialComments != null) {
```

```
                writeOrderComment(_commentID,_orderID,_specialComments);
            }
        }
        insertResultSet.close();
    }
```

For .NET, use the following code:

```
    public void WriteOrderComment(int comment_id,
        int order_id,
        string comments)
    {
        if (my_writer == null) {
            my_writer = new StreamWriter("c:\\MLdirect\
\orderComments.txt");
        }
        my_writer.WriteLine("{0}\t{1}\t{2}", comment_id, order_id,
comments);
        my_writer.Flush();
    }

    // Method for the handle_UploadData synchronization event.
    public void GetUpload(UploadData ut) {
        // Get UploadedTableData for remote table called OrderComments
        UploadedTableData order_comments_table_data =
            ut.GetUploadedTableByName("OrderComments");

        // Get inserts uploaded by the MobiLink client
        IDataReader new_comment_reader =
            order_comments_table_data.GetInserts();

        while (new_comment_reader.Read()) {
            // Columns are
            // 0 - "order_comment"
            // 1 - "comment_id"
            // 2 - "order_id"
            // You can look up these values using the DataTable returned
by:
            // order_comments_table_data.GetSchemaTable() if the send
            // column names option is turned on at the remote.
            // In this example, you just use the known column order to
            // determine the column indexes

            // Only process this insert if the order_comment is not null
            if (!new_comment_reader.IsDBNull(2)) {
                int comment_id = new_comment_reader.GetInt32(0);
                int order_id = new_comment_reader.GetInt32(1);
                string comments = new_comment_reader.GetString(2);
                WriteOrderComment(comment_id, order_id, comments);
            }
        }
        // Always close the reader when you are done with it!
        new_comment_reader.Close();
    }
```

6. Write the SetDownload method:

   a. Obtain a class instance representing the OrderComments table.

      Use the DBConnectionContext getDownloadData method to obtain a DownloadData instance. Use the DownloadData getDownloadTableByName method to return a DownloadTableData instance for the OrderComments table.

For Java, use the following code:

```
      public void SetDownload()
          throws SQLException, IOException
      {
          DownloadData download_d = _cc.getDownloadData();
          DownloadTableData download_td =
  download_d.getDownloadTableByName( "OrderComments" );
```

For .NET, use the following code:

```
      private const string read_file_path =
          "c:\\MLdirect\\orderResponses.txt";

      // Method for the handle_DownloadData synchronization event
      public void SetDownload() {
          if ((my_reader == null) && !File.Exists(read_file_path)) {
              System.Console.Out.Write("There is no file to read.");
              return;
          }
          DownloadTableData comments_for_download =

  _cc.GetDownloadData().GetDownloadTableByName("OrderComments");
```

Note
You create this table on the remote database in "Lesson 6: Set up your MobiLink client database" on page 167.

b. Obtain a prepared statement or IDbCommand that allows you to add insert or update operations to the download.

   For Java, use the DownloadTableData getUpsertPreparedStatement method to return a java.sql.PreparedStatement instance as follows:

```
          PreparedStatement update_ps =
  download_td.getUpsertPreparedStatement();
```

   For .NET, use the DownloadTableData GetUpsertCommand method as follows:

```
          // Add upserts to the set of operation that are going to be
          // applied at the remote database
          IDbCommand comments_upsert =
              comments_for_download.GetUpsertCommand();
```

c. Set the download data for each row.

   This code traverses through the *orderResponses.txt* and adds data to the MobiLink download.

   For Java, use the following code:

```
          // A BufferedReader for reading in responses
          if (my_reader == null)
              my_reader = new BufferedReader(new FileReader( "c:\\MLdirect
  \\orderResponses.txt"));

          // Get the next line from orderResponses
          String commentLine;
          commentLine = my_reader.readLine();

          // Send comment responses down to clients
          while (commentLine != null) {
```

```
                // Get the next line from orderResponses.txt
                String[] response_details = commentLine.split("\t");

                if (response_details.length != 3) {
                    System.err.println("Error reading from
orderResponses.txt");
                    System.err.println("Error setting direct row handling
download");
                    return;
                }
                int comment_id = Integer.parseInt(response_details[0]);
                int order_id = Integer.parseInt(response_details[1]);
                String updated_comment = response_details[2];

                // Set an order comment response in the MobiLink download
                update_ps.setInt(1, comment_id);
                update_ps.setInt(2, order_id);
                update_ps.setString(3, updated_comment);
                update_ps.executeUpdate();

                // Get next line
                commentLine = my_reader.readLine();
            }
```

For .NET, use the following code:

```
            if (my_reader == null) {
                my_reader = new StreamReader(read_file_path);
            }
            string comment_line;
            while ((comment_line = my_reader.ReadLine()) != null) {
                // Three values are on each line separated by '\t'
                string[] response_details = comment_line.Split('\t');
                if (response_details.Length != 3) {
                    throw (new SynchronizationException(
                        "Error reading from orderResponses.txt"));
                }
                int comment_id = System.Int32.Parse(response_details[0]);
                int order_id = System.Int32.Parse(response_details[1]);
                string comments = response_details[2];

                // Parameters of the correct number and type have
                // already been added so you just need to set the
                // values of the IDataParameter
                ((IDataParameter)(comments_upsert.Parameters[0])).Value =
                    comment_id;
                ((IDataParameter)(comments_upsert.Parameters[1])).Value =
                    order_id;
                ((IDataParameter)(comments_upsert.Parameters[2])).Value =
                    comments;
                // Add the upsert operation
                comments_upsert.ExecuteNonQuery();
            }
        }
```

d. Close the prepared statement used for adding insert or update operations to the download.

   For Java, use the following code:

```
            update_ps.close();
        }
```


For .NET, you do not need to close the IDbCommand. The object is destroyed automatically at the end of the download.

7. Write the EndDownload method.

   This method handles the end_download connection event and gives you an opportunity to free resources.

   For Java, use the following code:

```
    public void EndDownload()
        throws IOException
    {
        // Close i/o resources
        if (my_reader != null) my_reader.close();
        if (my_writer != null) my_writer.close();
    }
}
```

   For .NET, use the following code:

```
    public void EndDownload()
    {
        if (my_writer != null) {
            my_writer.Close();
            my_writer = null;
        }
        if (my_reader != null) {
            my_reader.Close();
            my_reader = null;
        }
    }
}
```

8. Save your code.

   For Java, save your code as *MobiLinkOrders.java* in your working directory *c:\MLdirect*.

   For .NET, save your code as *MobiLinkOrders.cs* in your working directory *c:\MLdirect*.

9. See "Complete MobiLinkOrders code listing (Java)" on page 162 or "Complete MobiLinkOrders code listing (.NET)" on page 164 to verify the code.

10. Compile your class file.

    a. Navigate to the directory containing your Java or .NET source files.

    b. Compile MobiLinkOrders and refer to the MobiLink server API library for Java or .NET.

       For Java, you need to reference *mlscript.jar*, located in *install-dir\java*.

       For Java, run the following command, replacing *C:\Program Files\SQL Anywhere 12* with your SQL Anywhere 12 directory:

```
javac -classpath "C:\Program Files\SQL Anywhere 12\java\mlscript.jar"
MobiLinkOrders.java
```

       For .NET, run the following command, replacing *C:\Program Files\SQL Anywhere 12* with your SQL Anywhere 12 directory:

```
csc /out:MobiLinkServerCode.dll /target:library /reference:"C:\Program
Files\SQL Anywhere 12\Assembly\v2\iAnywhere.MobiLink.Script.dll"
MobiLinkOrders.cs
```

Note
The example shown here does not ensure that Primary Key values are unique. See "Maintaining unique primary keys" [*MobiLink - Server Administration*].

Further reading

For more information about class constructors and DBConnectionContext, see:

- .NET synchronization logic: "Constructors" [*MobiLink - Server Administration*]
- Java synchronization logic: "Constructors" [*MobiLink - Server Administration*]

For more information about Java synchronization logic, see "Writing synchronization scripts in Java" [*MobiLink - Server Administration*].

For more information about .NET synchronization logic, see "Writing synchronization scripts in .NET" [*MobiLink - Server Administration*].

For more information about direct row handling, see "Direct row handling" [*MobiLink - Server Administration*].

Complete MobiLinkOrders code listing (Java)

The following is the complete MobiLinkOrders listing for Java direct row handling. For a step by step explanation, see "Lesson 4: Create a Java or .NET class for MobiLink direct row handling" on page 155.

```
import ianywhere.ml.script.*;
import java.io.*;
import java.sql.*;

public class MobiLinkOrders {

    // Class level DBConnectionContext
    DBConnectionContext _cc;

    // Java objects for file i/o
    FileWriter my_writer;
    BufferedReader my_reader;

    public MobiLinkOrders( DBConnectionContext cc )
        throws IOException, FileNotFoundException
    {
        // Declare a class-level DBConnectionContext
        _cc = cc;
    }

    public void writeOrderComment( int _commentID, int _orderID, String
_comments )
        throws IOException
    {
        if (my_writer == null)
            // A FileWriter for writing order comments
            my_writer = new FileWriter( "C:\\MLdirect\
```



```
\orderComments.txt",true);

        // Write out the order comments to remoteOrderComments.txt
        my_writer.write(_commentID + "\t" + _orderID + "\t" + _comments);
        my_writer.write( "\n" );
        my_writer.flush();
    }

    //  Method for the handle_UploadData synchronization event
    public void GetUpload( UploadData ut )
        throws SQLException, IOException
    {
        // Get an UploadedTableData for OrderComments
        UploadedTableData orderCommentsTbl =
ut.getUploadedTableByName("OrderComments");

        // Get inserts uploaded by the MobiLink client
        ResultSet insertResultSet = orderCommentsTbl.getInserts();

        while ( insertResultSet.next() )
        {
            // Get order comments
            int _commentID = insertResultSet.getInt("comment_id");
            int _orderID = insertResultSet.getInt("order_id");
            String _specialComments =
insertResultSet.getString("order_comment");
            if (_specialComments != null) {
                writeOrderComment(_commentID,_orderID,_specialComments);
            }
        }
        insertResultSet.close();
    }

    public void SetDownload()
        throws SQLException, IOException
    {
        DownloadData download_d = _cc.getDownloadData();

        DownloadTableData download_td =
download_d.getDownloadTableByName( "OrderComments" );

        PreparedStatement update_ps =
download_td.getUpsertPreparedStatement();
        // A BufferedReader for reading in responses
        if (my_reader == null)
            my_reader = new BufferedReader(new FileReader( "c:\\MLdirect\
\orderResponses.txt"));

        // Get the next line from orderResponses
        String commentLine;
        commentLine = my_reader.readLine();

        // Send comment responses down to clients
        while (commentLine != null) {
            // Get the next line from orderResponses.txt
            String[] response_details = commentLine.split("\t");

            if (response_details.length != 3) {
                System.err.println("Error reading from orderResponses.txt");
                System.err.println("Error setting direct row handling
download");
                return;
            }
            int comment_id = Integer.parseInt(response_details[0]);
```

```
                int order_id = Integer.parseInt(response_details[1]);
                String updated_comment = response_details[2];

                // Set an order comment response in the MobiLink download
                update_ps.setInt(1, comment_id);
                update_ps.setInt(2, order_id);
                update_ps.setString(3, updated_comment);
                update_ps.executeUpdate();

                // Get next line
                commentLine = my_reader.readLine();
            }
            update_ps.close();
        }

        public void EndDownload()
            throws IOException
        {
            // Close i/o resources
            if (my_reader != null) my_reader.close();
            if (my_writer != null) my_writer.close();
        }
    }
```

Complete MobiLinkOrders code listing (.NET)

The following is the complete MobiLinkOrders listing for .NET direct row handling. For a step by step explanation, see "Lesson 4: Create a Java or .NET class for MobiLink direct row handling" on page 155.

```
using iAnywhere.MobiLink.Script;
using System.IO;
using System.Data;
using System.Text;

public class MobiLinkOrders {
    // Class level DBConnectionContext
    private DBConnectionContext _cc = null;

    // Instances for file I/O
    private static StreamWriter my_writer = null;
    private static StreamReader my_reader = null;


    public MobiLinkOrders(DBConnectionContext cc) {
        _cc = cc;
    }

    public void WriteOrderComment(int comment_id,
        int order_id,
        string comments)
    {
        if (my_writer == null) {
            my_writer = new StreamWriter("c:\\MLdirect\\orderComments.txt");
        }
        my_writer.WriteLine("{0}\t{1}\t{2}", comment_id, order_id, comments);
        my_writer.Flush();
    }

    // Method for the handle_UploadData synchronization event.
    public void GetUpload(UploadData ut)
    {
```



```
        // Get UploadedTableData for remote table called OrderComments
        UploadedTableData order_comments_table_data =
            ut.GetUploadedTableByName("OrderComments");

        // Get inserts uploaded by the MobiLink client
        IDataReader new_comment_reader =
            order_comments_table_data.GetInserts();

        while (new_comment_reader.Read()) {
            // Columns are
            // 0 - "order_comment"
            // 1 - "comment_id"
            // 2 - "order_id"
            // You can look up these values using the DataTable returned by:
            // order_comments_table_data.GetSchemaTable() if the send
            // column names option is turned on at the remote.
            // In this example, you just use the known column order to
            // determine the column indexes

            // Only process this insert if the order_comment is not null
            if (!new_comment_reader.IsDBNull(2)) {
                int comment_id = new_comment_reader.GetInt32(0);
                int order_id = new_comment_reader.GetInt32(1);
                string comments = new_comment_reader.GetString(2);
                WriteOrderComment(comment_id, order_id, comments);
            }
        }
        // Always close the reader when you are done with it!
        new_comment_reader.Close();
    }

    private const string read_file_path =
        "c:\\MLdirect\\orderResponses.txt";

    // Method for the handle_DownloadData synchronization event
    public void SetDownload() {
        if ((my_reader == null) && !File.Exists(read_file_path)) {
            System.Console.Out.Write("There is no file to read.");
            return;
        }
        DownloadTableData comments_for_download =
            _cc.GetDownloadData().GetDownloadTableByName("OrderComments");

        // Add upserts to the set of operation that are going to be
        // applied at the remote database
        IDbCommand comments_upsert =
            comments_for_download.GetUpsertCommand();

        if (my_reader == null) {
            my_reader = new StreamReader(read_file_path);
        }
        string comment_line;
        while ((comment_line = my_reader.ReadLine()) != null) {
            // Three values are on each line separated by '\t'
            string[] response_details = comment_line.Split('\t');
            if (response_details.Length != 3) {
                throw (new SynchronizationException(
                    "Error reading from orderResponses.txt"));
            }
            int comment_id = System.Int32.Parse(response_details[0]);
            int order_id = System.Int32.Parse(response_details[1]);
            string comments = response_details[2];

            // Parameters of the correct number and type have
```

```
                // already been added so you just need to set the
                // values of the IDataParameter
                ((IDataParameter)(comments_upsert.Parameters[0])).Value =
                    comment_id;
                ((IDataParameter)(comments_upsert.Parameters[1])).Value =
                    order_id;
                ((IDataParameter)(comments_upsert.Parameters[2])).Value =
                    comments;
                // Add the upsert operation
                comments_upsert.ExecuteNonQuery();
            }
        }

        public void EndDownload()
        {
            if (my_writer != null) {
                my_writer.Close();
                my_writer = null;
            }
            if (my_reader != null) {
                my_reader.Close();
                my_reader = null;
            }
        }
    }
```

Lesson 5: Start the MobiLink server

In this lesson, you start the MobiLink server. You start the MobiLink server (mlsrv12) using the -c option to connect to your consolidated database. You use the -sl java or -sl dnet option to load your Java or .NET class, respectively.

To start the MobiLink server (mlsrv12)

- Connect to your consolidated database and load the class on the mlsrv12 command line.

  Replace *c:\MLdirect* with the location of your source files.

  For Java, run the following command:

  ```
  mlsrv12 -c "dsn=mldirect_db" -o serverOut.txt -v+ -dl -zu+ -x tcpip -sl
  java (-cp c:\MLdirect)
  ```

  For .NET, run the following command:

  ```
  mlsrv12 -c "dsn=mldirect_db;uid=DBA;pwd=sql" -o serverOut.txt -v+ -dl -zu+
  -x tcpip -sl dnet (-MLAutoLoadPath=c:\MLdirect)
  ```

  The MobiLink server messages window appears.

Below is a description of each MobiLink server option used in this tutorial. The options -o, -v, and -dl provide debugging and troubleshooting information. Using these logging options is appropriate in a development environment. For performance reasons, -v and -dl are typically not used in production.

Option	Description
-c	Precedes the connection string.
-o	Specifies the message log file *serverOut.txt*.
-v+	The -v option specifies what information is logged. Using -v+ sets maximum verbose logging.
-dl	Displays all log messages on screen.
-zu+	Adds new users automatically.
-x	Sets the communications protocol and parameters for MobiLink clients.
-sl java	Specifies a set of directories to search for class files, and forces the Java VM to load on server startup.
-sl dnet	Specifies the location of .NET assemblies and forces the CLR to load on server startup.

Further reading

For a complete list of MobiLink server options, see "MobiLink server options" [*MobiLink - Server Administration*].

For more information about loading Java and .NET classes, see "-sl java mlsrv12 option" [*MobiLink - Server Administration*] and "-sl dnet mlsrv12 option" [*MobiLink - Server Administration*], respectively.

Lesson 6: Set up your MobiLink client database

In this lesson, you use a SQL Anywhere database for your consolidated database and your MobiLink client. For tutorial purposes your MobiLink client, consolidated database, and MobiLink server all reside on the same computer.

To set up the MobiLink client database, you create the RemoteOrders and OrderComments tables. The RemoteOrders table corresponds to the RemoteOrders table on the consolidated database. The MobiLink server uses SQL-based scripts to synchronize remote orders. The OrderComments table is only used on client databases. The MobiLink server processes the OrderComments tables using special events.

You create a synchronization user, publication, and subscription on the client database after creating the tables. Publications identify the tables and columns on your remote database that you want synchronized. These tables and columns are called **articles**. A synchronization subscription subscribes a MobiLink user to a publication.

To set up your MobiLink client database

1. Create your MobiLink client database using the dbinit command line utility.

   Run the following command:

   ```
   dbinit -i -k remote1
   ```

The -i and -k options omit jConnect support and Watcom SQL compatibility views, respectively.

2. Start your MobiLink client database using the dbeng12 command line utility.

   Run the following command:

   ```
   dbeng12 remote1
   ```

3. Connect to your MobiLink client database using Interactive SQL.

   Run the following command:

   ```
   dbisql -c "server=remote1;uid=DBA;pwd=sql"
   ```

4. Create the RemoteOrders table.

   Run the following SQL script in Interactive SQL:

   ```
   CREATE TABLE RemoteOrders (
       order_id           INTEGER NOT NULL,
       product_id         INTEGER NOT NULL,
       quantity           INTEGER,
       order_status       VARCHAR(10) DEFAULT 'new',
       PRIMARY KEY(order_id)
   )
   ```

5. Create the OrderComments table.

   Run the following SQL script in Interactive SQL:

   ```
   CREATE TABLE OrderComments (
       comment_id         INTEGER NOT NULL,
       order_id           INTEGER NOT NULL,
       order_comment      VARCHAR(255),
       PRIMARY KEY(comment_id),
       FOREIGN KEY(order_id) REFERENCES RemoteOrders(order_id)
   )
   ```

6. Create your MobiLink synchronization user, publication, and subscription.

   Run the following SQL script in Interactive SQL:

   ```
   CREATE SYNCHRONIZATION USER ml_sales1;
   CREATE PUBLICATION order_publ (TABLE RemoteOrders, TABLE OrderComments);
   CREATE SYNCHRONIZATION SUBSCRIPTION TO order_publ FOR ml_sales1
    TYPE TCPIP ADDRESS 'host=localhost'
   ```

   Note
   You specify how to connect to the MobiLink server using the TYPE and ADDRESS clauses in the CREATE SYNCHRONIZATION SUBSCRIPTION statement.

   You can use publications to determine what data is synchronized. In this case you specify the entire RemoteOrders and OrderComments tables.

Stay connected in Interactive SQL for the next lesson.

Further reading

For information about creating a SQL Anywhere database, see "Initialization utility (dbinit)" [*SQL Anywhere Server - Database Administration*].

For information about MobiLink clients, see "MobiLink clients" [*MobiLink - Client Administration*].

For information about creating MobiLink objects on the client, see:

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

Lesson 7: Synchronize

The dbmlsync utility initiates MobiLink synchronization for SQL Anywhere remote databases. Before starting dbmlsync, add order data and comments to your remote database.

To set up your remote data (client-side)

1. Connect to the MobiLink client database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "server=remote1;uid=DBA;pwd=sql"
   ```

2. Add an order to the RemoteOrders table in the client database.

   Run the following SQL script in Interactive SQL:

   ```
   INSERT INTO RemoteOrders (order_id, product_id, quantity, order_status)
    VALUES (1,12312,10,'new')
   ```

3. Add a comment to the OrderComments table in the client database.

   Run the following SQL script in Interactive SQL:

   ```
   INSERT INTO OrderComments (comment_id, order_id, order_comment)
    VALUES (1,1,'send promotional material with the order')
   ```

4. Commit your changes.

   Run the following SQL script in Interactive SQL:

   ```
   COMMIT;
   ```

To start the synchronization client (client-side)

- Run the following command at the command prompt:

```
dbmlsync -c "server=remote1;uid=DBA;pwd=sql" -e scn=on -o rem1.txt -v+
```

The following table contains a description for each dbmlsync option used in this lesson:

Option	Description
-c	Specifies the connection string.
-e scn	Sets SendColumnNames to on. This is required by direct row handling if you want to reference columns by name.
-o	Specifies the message log file *rem1.txt*.
-v+	The -v option specifies what information is logged. Using -v+ sets maximum verbose logging.

Once you have started the MobiLink synchronization client, an output screen appears indicating that the synchronization succeeded. SQL-based synchronization transferred rows in the client RemoteOrders table to the RemoteOrders table in the consolidated database.

Java or .NET processing inserted your comment in *orderComments.txt*. In the next procedure, you insert a response in *orderResponses.txt* to download to the remote database.

To return comments using direct row handling downloads (server-side and client-side)

1. Close any SQL Anywhere MobiLink Client windows
2. Insert return comments. This action takes place on the server side.

   Add the following text to *orderResponses.txt*. You must separate entries using the tab character. At the end of the line, press Enter.

   ```
   1  1  Promotional material shipped
   ```

3. Run synchronization using the dbmlsync client utility.

   This action takes place on the client-side.

   Run the following command:

   ```
   dbmlsync -c "server=remote1;uid=DBA;pwd=sql" -o rem1.txt -v+ -e scn=on
   ```

   The MobiLink client utility appears.

   In Interactive SQL, select from the OrderComments table to verify that the row was downloaded.

Note
Rows downloaded using direct row handling are not printed by the mlsrv12 -v+ option, but are printed in the remote log by the remote -v+ option.

Further reading

For more information about dbmlsync, see "SQL Anywhere clients" [*MobiLink - Client Administration*].

Cleanup

Remove tutorial materials from your computer.

To remove tutorial materials

1. Close all instances of Interactive SQL.
2. Close the SQL Anywhere, MobiLink, and synchronization client windows.
3. Delete all tutorial-related DSNs:
   a. Start the ODBC Administrator.

      Type the following command at the command prompt:

      ```
      odbcad32
      ```

   b. Remove the **mldirect_db** data source.
4. Delete the consolidated and remote databases:
   a. Navigate to the directory containing your consolidated and remote databases.
   b. Delete *MLconsolidated.db*, *MLconsolidated.log*, *remote1.db*, and *remote1.log*.

Further reading

For more information about MobiLink server APIs, see:

- "Writing synchronization scripts in Java" [*MobiLink - Server Administration*]
- "Writing synchronization scripts in .NET" [*MobiLink - Server Administration*]

For more information about MobiLink synchronization, see "Understanding MobiLink synchronization" on page 1.

For more information about MobiLink clients, see "MobiLink clients" [*MobiLink - Client Administration*].

For more information about MobiLink direct row handling, see "Direct row handling" [*MobiLink - Server Administration*].

Tutorial: Synchronizing with Microsoft Excel

Introduction to synchronizing with Excel

This tutorial guides you through the basic steps for using direct row handling to synchronize data in a Microsoft Excel spreadsheet with MobiLink clients. This tutorial uses a Java implementation as an example.

This tutorial shows you how to implement MobiLink direct row handling so that you can use a data source other than a supported consolidated database.

You can use direct row handling to communicate remote data to any central data source, application, or web service.

For more information about direct row handling, see "Direct row handling" [*MobiLink - Server Administration*].

This tutorial shows you how to synchronize data in a Microsoft Excel spreadsheet to remote clients.

Required software

- SQL Anywhere 12
- Java Software Development Kit
- Microsoft Office Excel 2007 or later

Competencies and experience

You require:

- Familiarity with Java
- Familiarity with Microsoft Excel
- Basic knowledge of MobiLink event scripts and MobiLink synchronization

Goals

You gain competence and familiarity with:

- The MobiLink server API for Java
- Creating methods for MobiLink direct row handling
- Accessing data from a Microsoft Excel worksheet using Java

Suggested background reading

- "Understanding MobiLink synchronization" on page 1
- "Synchronization techniques" [*MobiLink - Server Administration*]
- "Direct row handling" [*MobiLink - Server Administration*]

MobiLink code exchange samples are located at http://www.sybase.com/detail?id=1058600#319.

You can post questions on the MobiLink newsgroup: sybase.public.sqlanywhere.mobilink.

Lesson 1: Set up an Excel worksheet

In this lesson, you create an Excel worksheet and use the Microsoft Excel Driver to define an ODBC data source. The Excel worksheet stores product information.

To set up an Excel data source

1. Open Microsoft Excel and create a new workbook.
2. In the default worksheet, add the following contents under the respective **A**, **B**, **C** column headers:

comment_id	order_id	order_comment
2	1	Promotional material shipped
3	1	More information about material required

3. Change the default worksheet name **Sheet1** to **order_sheet**.
   a. Double-click the **Sheet1** tab.
   b. Type **order_sheet**.
4. Save the Excel workbook.

   This tutorial assumes *c:\MLobjexcel* as the working directory for server-side components. Save the workbook as *order_central.xlsx* in this working directory.
5. Use the Microsoft Excel Driver to create an ODBC data source:
   a. Click **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **ODBC Data Source Administrator**.
   b. Click the **User DSN** tab.
   c. Click **Add**.
   d. Click **Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.xlsb)**.
   e. Click **Finish**.
   f. In the **Data Source Name** field, type **excel_datasource**.
   g. Click **Select Workbook** and browse to *c:\MLobjexcel\order_central.xlsx*, the file containing your worksheet.
   h. Uncheck the **Read Only** option.
   i. Click **OK** on all open ODBC Data Source Administrator windows.

Lesson 2: Set up your MobiLink consolidated database

Your MobiLink consolidated database is a central repository of data and includes MobiLink system tables and stored procedures used to manage the synchronization process. With direct row handling, you

synchronize with a data source other than a consolidated database, but you still need a consolidated database to maintain information used by the MobiLink server.

In this lesson, you:

- Create a database and define an ODBC data source.
- Add data tables to synchronize to remote clients.
- Install MobiLink system tables and stored procedures.

Note
If you already have a MobiLink consolidated database set up with MobiLink system objects and a DSN, you can skip this lesson.

Create your consolidated database

In this tutorial, you create a SQL Anywhere database using the Sybase Central **Create Database Wizard**.

To create your SQL Anywhere RDBMS

1. Click **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.
2. Click **Tools** » **SQL Anywhere 12** » **Create Database**.
3. Click **Next**.
4. Leave the default of **Create Database On This Computer**, and then click **Next**.
5. In the **Save The Main Database File To The Following File** field, type the file name and path for the database. For example, *c:\MLobjexcel\MLconsolidated.db*.
6. Follow the remaining instructions in the **Create Database Wizard** and accept the default values. On the **Connect To The Database** page, clear the **Stop The Database After Last Disconnect** option.
7. Click **Finish**.

   The MLconsolidated database appears in Sybase Central.
8. Click **Close** on the **Creating Database** window.

Define an ODBC data source for the consolidated database

Use the SQL Anywhere 12 driver to define an ODBC data source for the MLconsolidated database.

To define an ODBC data source for the consolidated database

1. From the Sybase Central **Tools** menu, choose **SQL Anywhere 12** » **Open ODBC Administrator**.
2. Click the **User DSN** tab, and click **Add**.
3. In the **Create New Data Source** window, click **SQL Anywhere 12** and click **Finish**.

4. Perform the following tasks in the **ODBC Configuration For SQL Anywhere** window:
   a. Click the **ODBC** tab.
   b. In the **Data Source Name** field, type **mlexcel_db**.
   c. Click the **Login** tab.
   d. In the **User ID** field, type **DBA**.
   e. In the **Password** field, type **sql**.
   f. From the **Action** dropdown list, choose **Connect To A Running Database On This Computer**.
   g. In the **Server Name** field, type **MLconsolidated**.
   h. Click **OK**.
5. Close ODBC data source administrator.

   Click **OK** on the **ODBC Data Source Administrator** window.

Create tables for synchronization

In this procedure, you create the RemoteOrders table in the MobiLink consolidated database. The RemoteOrders table contains the following columns:

Column	Description
order_id	A unique identifier for orders.
product_id	A unique identifier for products.
quantity	The number of items sold.
order_status	The order status.
last_modified	The last modification date of a row. You use this column for timestamp based downloads, a common technique used to filter rows for efficient synchronization.

To create the RemoteOrders table

1. Connect to your database using Interactive SQL.

   You can start Interactive SQL from Sybase Central or from a command prompt.

   - To start Interactive SQL from Sybase Central, right-click the **MLconsolidated - DBA** database and click **Open Interactive SQL**.
   - To start Interactive SQL at a command prompt, run the following command:

     ```
     dbisql -c "dsn=mlexcel_db"
     ```

2. Run the following SQL script in Interactive SQL to create the RemoteOrders table.

   ```
   CREATE TABLE RemoteOrders (
       order_id           INTEGER NOT NULL,
   ```

```
    product_id          INTEGER NOT NULL,
    quantity            INTEGER,
    order_status        VARCHAR(10) DEFAULT 'new',
    last_modified       TIMESTAMP DEFAULT CURRENT TIMESTAMP,
    PRIMARY KEY(order_id)
)
```

Interactive SQL creates the RemoteOrders table in your consolidated database.

Stay connected in Interactive SQL for the next procedure.

Run MobiLink setup scripts

You can find setup scripts for each supported consolidated database in the *MobiLink/setup* subdirectory of your SQL Anywhere 12 installation.

In this procedure you set up a SQL Anywhere consolidated database. You do this using the *syncsa.sql* setup script. Running *syncsa.sql* creates a series of system tables and stored procedures prefaced with **ml_**. The MobiLink server works with these tables and stored procedures in the synchronization process.

To install MobiLink system tables

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "dsn=mlexcel_db"
   ```

2. Run the following command in Interactive SQL to create MobiLink system tables and stored procedures. Replace *C:\Program Files\SQL Anywhere 12* with the location of your SQL Anywhere 12 installation.

   ```
   READ "C:\Program Files\SQL Anywhere 12\MobiLink\setup\syncsa.sql"
   ```

   Interactive SQL applies *syncsa.sql* to your consolidated database.

Stay connected in Interactive SQL for the next lesson.

Further reading

For information about creating SQL Anywhere databases, see "Initialization utility (dbinit)" [*SQL Anywhere Server - Database Administration*].

For information about creating tables, see "CREATE TABLE statement" [*SQL Anywhere Server - SQL Reference*].

For information about setting up MobiLink consolidated databases, see "MobiLink consolidated databases" [*MobiLink - Server Administration*].

Lesson 3: Add synchronization scripts

In this lesson you add scripts to your consolidated database for SQL row handling and direct row handling.

SQL row handling

SQL row handling allows you to synchronize remote data with tables in your MobiLink consolidated database. SQL-based scripts define:

- How data that is uploaded from a MobiLink client is to be applied to the consolidated database.
- What data should be downloaded from the consolidated database.

In this lesson, you write synchronization scripts for the following SQL-based upload and download events.

- **upload_insert** This event defines how new orders inserted in a client database should be applied to the consolidated database.
- **download_cursor** This event defines the orders that should be downloaded to remote clients.
- **download_delete_cursor** This event is required when using synchronization scripts that are not upload-only. You set the MobiLink server to ignore this event for the purpose of this tutorial.

In this procedure, you add synchronization script information to your MobiLink consolidated database using stored procedures.

To add SQL-based scripts to MobiLink system tables

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "dsn=mlexcel_db"
   ```

2. Use the ml_add_table_script stored procedure to add SQL-based table scripts for the upload_insert, download_cursor and download_delete_cursor events.

   Run the following SQL script in Interactive SQL. The upload_insert script inserts the uploaded order_id, product_id, quantity, and order_status into the MobiLink consolidated database. The download_cursor script uses timestamp-based filtering to download updated rows to remote clients.

   ```
   CALL ml_add_table_script( 'default', 'RemoteOrders',
       'upload_insert',
       'INSERT INTO RemoteOrders( order_id, product_id, quantity,
   order_status)
       VALUES( ?, ?, ?, ? )' );

   CALL ml_add_table_script( 'default', 'RemoteOrders',
       'download_cursor',
       'SELECT order_id, product_id, quantity, order_status
       FROM RemoteOrders WHERE last_modified >= ?');

   CALL ml_add_table_script( 'default', 'RemoteOrders',
       'download_delete_cursor', '--{ml_ignore}');

   COMMIT
   ```

Direct row handling processing

In this tutorial, you use direct row handling to add special processing to a SQL-based synchronization system. In this procedure you register method names corresponding to the handle_UploadData,

handle_DownloadData, download_cursor, and download_delete_cursor events. You create your own Java class in “Lesson 4: Create a Java class for MobiLink direct row handling” on page 179.

To add information for direct row handling in MobiLink system tables

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "dsn=mlexcel_db"
   ```

2. Register Java methods for the handle_UploadData and handle_DownloadData events.

   Run the following SQL script in Interactive SQL:

   ```
   CALL ml_add_java_connection_script( 'default',
       'handle_UploadData',
       'MobiLinkOrders.GetUpload' );

   CALL ml_add_java_connection_script( 'default',
       'handle_DownloadData',
       'MobiLinkOrders.SetDownload' );
   ```

   Interactive SQL registers the GetUpload and SetDownload methods for the handle_UploadData and handle_DownloadData events, respectively. You create these methods in an upcoming lesson.

3. Register the download_cursor and download_delete_cursor events.

   Run the following SQL script in Interactive SQL:

   ```
   CALL ml_add_table_script( 'default', 'OrderComments',
       'download_cursor', '--{ml_ignore}');

   CALL ml_add_table_script( 'default', 'OrderComments',
       'download_delete_cursor', '--{ml_ignore}');
   ```

   The download_cursor and download_delete_cursor events must be registered for the OrderComments table when using scripts because the synchronization is bi-directional and not upload-only. See “Required scripts” [*MobiLink - Server Administration*].

4. Commit your changes.

   Run the following SQL script in Interactive SQL:

   ```
   COMMIT;
   ```

You can now close Interactive SQL.

Further reading

For information about using SQL-based events to upload data from remote clients to a MobiLink consolidated database, see:

- "Writing scripts to upload rows" [*MobiLink - Server Administration*]
- "upload_insert table event" [*MobiLink - Server Administration*]
- "upload_update table event" [*MobiLink - Server Administration*]
- "upload_delete table event" [*MobiLink - Server Administration*]

For information about uploading data to data sources other than consolidated databases, see "Handling direct uploads" [*MobiLink - Server Administration*].

For information about using SQL-based events to download data from a MobiLink consolidated database, see:

- "Writing scripts to download rows" [*MobiLink - Server Administration*]
- "download_cursor table event" [*MobiLink - Server Administration*]
- "download_delete_cursor table event" [*MobiLink - Server Administration*]

For information about downloading data to data sources other than consolidated databases, see "Handling direct downloads" [*MobiLink - Server Administration*].

For information about the synchronization event sequence, see "Overview of MobiLink events" [*MobiLink - Server Administration*].

For information about synchronization techniques for download filtering, see "Timestamp-based downloads" [*MobiLink - Server Administration*] and "Partitioning rows among remote databases" [*MobiLink - Server Administration*].

For information about managing scripts, see "Adding and deleting scripts" [*MobiLink - Server Administration*].

For information about direct row handling, see "Direct row handling" [*MobiLink - Server Administration*].

Lesson 4: Create a Java class for MobiLink direct row handling

In this lesson, you use direct row handling to process rows in the OrderComments table in your client database. You add the following methods for direct row handling:

- **GetUpload** You use this method for the handle_UploadData event. GetUpload writes uploaded comments to the excel worksheet *order_central.xlsx*.

- **SetDownload** You use this method for the handle_DownloadData event. SetDownload retrieves the data stored in the excel worksheet *order_central.xlsx* and sends it to remote clients.

The following procedure shows you how to create a Java class including your methods for processing. For a complete listing, see "Complete MobiLinkOrders code listing (Java)" on page 183.

To create a Java class for download-only direct row handling

1. Start writing a new class named MobiLinkOrders.

   Write the following code:

   ```
   import ianywhere.ml.script.*;
   import java.io.*;
   import java.sql.*;

   public class MobiLinkOrders {
   ```

2. Declare a class-level DBConnectionContext instance.

   Append the following code:

   ```
       // Class level DBConnectionContext
       DBConnectionContext _cc;
   ```

   The MobiLink server passes a DBConnectionContext instance to your class constructor. DBConnectionContext encapsulates information about the current connection with the MobiLink consolidated database.

3. Create your class constructor.

   Your class constructor sets your class-level DBConnectionContext instance.

   Append the following code:

   ```
       public MobiLinkOrders( DBConnectionContext cc ) {
           throws IOException, FileNotFoundException {
           // Declare a class-level DBConnectionContext
           _cc = cc;
       }
   ```

4. Write the GetUpload method.

   The GetUpload method obtains an UploadedTableData class instance representing the OrderComments table. The OrderComments table contains special comments made by remote sales employees. You create this table in "Lesson 6: Set up your MobiLink client database" on page 185. The UploadedTableData getInserts method returns a result set for new order comments.

   Append the following code:

   ```
       //  Method for the handle_UploadData synchronization event
      public void GetUpload( UploadData ut ) throws SQLException, IOException
   {

           //  Get an UploadedTableData for OrderComments
           UploadedTableData orderCommentsTbl =
   ut.getUploadedTableByName("OrderComments");

           // Get inserts uploaded by the MobiLink client
           ResultSet insertResultSet = orderCommentsTbl.getInserts();

           try {
               // Connect to the excel worksheet through ODBC
   ```



```
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con =
DriverManager.getConnection( "jdbc:odbc:excel_datasource" );

            while( insertResultSet.next() ) {
                // Get order comments
                int _commentID = insertResultSet.getInt("comment_id");
                int _orderID = insertResultSet.getInt("order_id");
                String _specialComments =
insertResultSet.getString("order_comment");

               // Execute an insert statement to add the order comment to
the worksheet
                PreparedStatement st = con.prepareStatement("INSERT INTO
[order_sheet$]"
                    + "(order_id, comment_id, order_comment) VALUES
(?,?,?)" );
                st.setString( 1, Integer.toString(_orderID) );
                st.setString( 2, Integer.toString(_commentID) );
                st.setString( 3, _specialComments );
                st.executeUpdate();
                st.close();
            }
            con.close();
        } catch(Exception ex) {
            System.err.print("Exception: ");
            System.err.println(ex.getMessage());
        }
        insertResultSet.close();
    }
```

5. Write the SetDownload method:

   a. Obtain a class instance representing the OrderComments table.

      Use the DBConnectionContext getDownloadData method to obtain a DownloadData instance. Use the DownloadData getDownloadTableByName method to return a DownloadTableData instance for the OrderComments table.

      Append the following code:

```
    public void SetDownload() throws SQLException, IOException {
        DownloadData download_d = _cc.getDownloadData();
        DownloadTableData download_td =
  download_d.getDownloadTableByName( "OrderComments" );
```

      Note
      You create this table on the remote database in "Lesson 6: Set up your MobiLink client database" on page 185.

   b. Obtain a prepared statement or IDbCommand that allows you to add insert or update operations to the download.

      Use the DownloadTableData getUpsertPreparedStatement method to return a java.sql.PreparedStatement instance.

      Append the following code:

```
        // Prepared statement to compile upserts (inserts or updates).
        PreparedStatement download_upserts =
  download_td.getUpsertPreparedStatement();
```

c. Set the download data for each row.

The following code traverses through the *order_central.xlsx* worksheet and adds data to the MobiLink download.

Append the following code:

```
            try {
                // Connect to the excel worksheet through ODBC
                Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
                Connection con =
    DriverManager.getConnection( "jdbc:odbc:excel_datasource" );

                // Retrieve all the rows in the worksheet
                Statement st = con.createStatement();
                ResultSet Excel_rs = st.executeQuery( "SELECT * FROM
    [order_sheet$]" );

                while (Excel_rs.next()) {
                    // Retrieve the row data
                    int Excel_comment_id = Excel_rs.getInt(1);
                    int Excel_order_id = Excel_rs.getInt(2);
                    String Excel_comment = Excel_rs.getString(3);

                    // Add the Excel data to the MobiLink download.
                    download_upserts.setInt( 1, Excel_comment_id );
                    download_upserts.setInt( 2, Excel_order_id );
                    download_upserts.setString( 3, Excel_comment );
                    download_upserts.executeUpdate();
                }
                // close the excel result set, statement, and connection.
                Excel_rs.close();
                st.close();
                con.close();
            } catch(Exception ex) {
                System.err.print("Exception: ");
                System.err.println(ex.getMessage());
            }
```

d. Close the prepared statement used for adding insert or update operations to the download, end the method and the class.

Append the following code:

```
            download_upserts.close();
        }
    }
```

6. Save your Java code as *MobiLinkOrders.java* in your working directory *c:\MLobjexcel*.

See "Complete MobiLinkOrders code listing (Java)" on page 183 to verify the code in *MobiLinkOrders.java*.

7. Compile your class file.

a. Navigate to the directory containing your Java source files.

b. Compile MobiLinkOrders that refer to the MobiLink server API library for Java.

You need to reference *mlscript.jar*, located in *install-dir\Java*.

Run the following command, replacing *C:\Program Files\SQL Anywhere 12* with your SQL Anywhere 12 directory:


```
javac -classpath "C:\Program Files\SQL Anywhere 12\java\mlscript.jar"
MobiLinkOrders.java
```

Further reading

For more information about synchronization logic, see "Writing synchronization scripts in Java" [*MobiLink - Server Administration*].

For more information about direct row handling, see "Direct row handling" [*MobiLink - Server Administration*].

Complete MobiLinkOrders code listing (Java)

The following listing shows the complete Java MobiLinkOrders class code used for this tutorial. For a step by step explanation, see "Lesson 4: Create a Java class for MobiLink direct row handling" on page 179.

```
import ianywhere.ml.script.*;
import java.io.*;
import java.sql.*;

public class MobiLinkOrders {

    // Class level DBConnectionContext
    DBConnectionContext _cc;

    public MobiLinkOrders( DBConnectionContext cc )
        throws IOException, FileNotFoundException {
        // Declare a class-level DBConnectionContext
        _cc = cc;
    }

    //  Method for the handle_UploadData synchronization event
    public void GetUpload( UploadData ut )
        throws SQLException, IOException {
        // Get an UploadedTableData for OrderComments
        UploadedTableData orderCommentsTbl =
ut.getUploadedTableByName("OrderComments");

        // Get inserts uploaded by the MobiLink client
        ResultSet insertResultSet = orderCommentsTbl.getInserts();

        try {
            // Connect to the excel worksheet through ODBC
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con =
DriverManager.getConnection( "jdbc:odbc:excel_datasource" );

            while( insertResultSet.next() ) {
                // Get order comments
                int _commentID = insertResultSet.getInt("comment_id");
                int _orderID = insertResultSet.getInt("order_id");
                String _specialComments =
insertResultSet.getString("order_comment");

              // Execute an insert statement to add the order comment to the
worksheet
                PreparedStatement st = con.prepareStatement("INSERT INTO
[order_sheet$]"
                    + "(order_id, comment_id, order_comment) VALUES
```

```
(?,?,?)" );
                st.setString( 1, Integer.toString(_orderID) );
                st.setString( 2, Integer.toString(_commentID) );
                st.setString( 3, _specialComments );
                st.executeUpdate();
                st.close();
            }
            con.close();
        } catch(Exception ex) {
            System.err.print("Exception: ");
            System.err.println(ex.getMessage());
        }
        insertResultSet.close();
    }

    public void SetDownload() throws SQLException, IOException {
        DownloadData download_d = _cc.getDownloadData();
        DownloadTableData download_td =
download_d.getDownloadTableByName( "OrderComments" );

        // Prepared statement to compile upserts (inserts or updates).
        PreparedStatement download_upserts =
download_td.getUpsertPreparedStatement();

        try {
            // Connect to the excel worksheet through ODBC
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con =
DriverManager.getConnection( "jdbc:odbc:excel_datasource" );

            // Retrieve all the rows in the worksheet
            Statement st = con.createStatement();
            ResultSet Excel_rs = st.executeQuery( "select * from [order_sheet
$]" );

            while (Excel_rs.next()) {
                // Retrieve the row data
                int Excel_comment_id = Excel_rs.getInt(1);
                int Excel_order_id = Excel_rs.getInt(2);
                String Excel_comment = Excel_rs.getString(3);

                // Add the Excel data to the MobiLink download.
                download_upserts.setInt( 1, Excel_comment_id );
                download_upserts.setInt( 2, Excel_order_id );
                download_upserts.setString( 3, Excel_comment );
                download_upserts.executeUpdate();
            }

            // Close the excel result set, statement, and connection.
            Excel_rs.close();
            st.close();
            con.close();
        } catch (Exception ex) {
            System.err.print("Exception: ");
            System.err.println(ex.getMessage());
        }
        download_upserts.close();
    }
}
```


Lesson 5: Start the MobiLink server

In this lesson, you start the MobiLink server. You start the MobiLink server (mlsrv12) using the -c option to connect to your consolidated database, and the -sl java option to load your Java class.

To start the MobiLink server for direct row handling

- Connect to your consolidated database and load the class on the mlsrv12 command line.

  Run the following command. Replace *c:\MLobjexcel* with the location of your Java source files.

  ```
  mlsrv12 -c "dsn=mlexcel_db" -o serverOut.txt -v+ -dl -zu+ -x tcpip  -sl
  java (-cp c:\MLobjexcel)
  ```

  The MobiLink server messages window appears.

Below is a description of each MobiLink server option used in this tutorial. The options -o, -v, and -dl provide debugging and troubleshooting information. Using these logging options is appropriate in a development environment. For performance reasons, -v and -dl are typically not used in production.

Option	Description
-c	Precedes the connection string.
-o	Specifies the message log file *serverOut.txt*.
-v+	The -v option specifies what information is logged. Using -v+ sets maximum verbose logging.
-dl	Displays all log messages on screen.
-zu+	Adds new users automatically.
-x	Sets the communications protocol and parameters for MobiLink clients.
-sl java	Specifies a set of directories to search for class files, and forces the Java VM to load on server startup.

Further reading

For a complete list of MobiLink server options, see "MobiLink server options" [*MobiLink - Server Administration*].

For more information about loading Java classes, see "-sl java mlsrv12 option" [*MobiLink - Server Administration*].

Lesson 6: Set up your MobiLink client database

In this lesson, you use a SQL Anywhere database for your consolidated database and your MobiLink client. For tutorial purposes your MobiLink client, consolidated database, and MobiLink server all reside on the same computer.

To set up the MobiLink client database, you create the RemoteOrders and OrderComments tables. The RemoteOrders table corresponds to the RemoteOrders table on the consolidated database. The MobiLink server uses SQL-based scripts to synchronize remote orders. The OrderComments table is only used on client databases. The MobiLink server processes the OrderComments tables using special events.

You also create a synchronization user, publication, and subscription on the client database.

To set up your MobiLink client database

1. Create your MobiLink client database using the dbinit command line utility.

   Run the following command:

   ```
   dbinit -i -k remote1
   ```

   The -i and -k options omit jConnect support and Watcom SQL compatibility views, respectively.

2. Start your MobiLink client database using the dbeng12 command line utility.

   Run the following command:

   ```
   dbeng12 remote1
   ```

3. Connect to your MobiLink client database using Interactive SQL.

   Run the following command:

   ```
   dbisql -c "server=remote1;uid=DBA;pwd=sql"
   ```

4. Create the RemoteOrders table.

   Run the following SQL script in Interactive SQL:

   ```
   CREATE TABLE RemoteOrders (
       order_id           INTEGER NOT NULL,
       product_id         INTEGER NOT NULL,
       quantity           INTEGER,
       order_status       VARCHAR(10) DEFAULT 'new',
       PRIMARY KEY(order_id)
   )
   ```

5. Create the OrderComments table.

   Run the following SQL script in Interactive SQL:

   ```
   CREATE TABLE OrderComments (
       comment_id         INTEGER NOT NULL,
       order_id           INTEGER NOT NULL,
       order_comment      VARCHAR(255),
       PRIMARY KEY(comment_id),
       FOREIGN KEY(order_id) REFERENCES RemoteOrders(order_id)
   )
   ```

6. Create your MobiLink synchronization user, publication, and subscription.

   Run the following SQL script in Interactive SQL:


```
CREATE SYNCHRONIZATION USER ml_sales1;
CREATE PUBLICATION order_publ (TABLE RemoteOrders, TABLE OrderComments);
CREATE SYNCHRONIZATION SUBSCRIPTION TO order_publ FOR ml_sales1
 TYPE TCPIP ADDRESS 'host=localhost'
```

Note
You specify how to connect to the MobiLink server using the TYPE and ADDRESS clauses in the CREATE SYNCHRONIZATION SUBSCRIPTION statement.

You can use publications to determine what data is synchronized. In this case you specify the entire RemoteOrders and OrderComments tables.

Stay connected in Interactive SQL for the next lesson.

Further reading

For information about creating a SQL Anywhere database, see "Initialization utility (dbinit)" [*SQL Anywhere Server - Database Administration*].

For information about MobiLink clients, see "MobiLink clients" [*MobiLink - Client Administration*].

For information about creating MobiLink objects on the client, see:

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

Lesson 7: Synchronize

The dbmlsync utility initiates MobiLink synchronization for SQL Anywhere remote databases. Before starting dbmlsync, add order data and comments to your remote database.

To set up your remote data (client-side)

1. Connect to the MobiLink client database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "server=remote1;uid=DBA;pwd=sql"
   ```

2. Add an order to the RemoteOrders table in the client database.

   Run the following SQL script in Interactive SQL:

   ```
   INSERT INTO RemoteOrders (order_id, product_id, quantity, order_status)
    VALUES (1,12312,10,'new')
   ```

3. Add a comment to the OrderComments table in the client database.

Run the following SQL script in Interactive SQL:

```
INSERT INTO OrderComments (comment_id, order_id, order_comment)
 VALUES (1,1,'send promotional material with the order')
```

4. Commit your changes.

   Run the following SQL script in Interactive SQL:

   ```
   COMMIT;
   ```

To start the synchronization client (client-side)

- Run the following command at the command prompt:

  ```
  dbmlsync -c "server=remote1;uid=DBA;pwd=sql" -e scn=on -o rem1.txt -v+
  ```

  The following table contains a description for each dbmlsync option used in this lesson:

Option	Description
-c	Specifies the connection string.
-e scn	Sets SendColumnNames to on. This is required by direct row handling if you want to reference columns by name.
-o	Specifies the message log file *rem1.txt*.
-v+	The -v option specifies what information is logged. Using -v+ sets maximum verbose logging.

Once you have started the MobiLink synchronization client, an output screen appears indicating that the synchronization succeeded. SQL-based synchronization transferred rows in the client RemoteOrders table to the RemoteOrders table in the consolidated database.

Java processing inserted your comment in the *order_central.xlsx* worksheet. The information stored in the *order_central.xlsx* worksheet is downloaded to the client.

In Interactive SQL, select from the OrderComments table to verify that the row was downloaded.

Note
Rows downloaded using direct row handling are not printed by the mlsrv12 -v+ option, but are printed in the remote log by the remote -v+ option.

Further reading

For more information about dbmlsync, see "SQL Anywhere clients" [*MobiLink - Client Administration*].

Cleanup

Remove tutorial materials from your computer.

To remove tutorial materials from your computer

1. Close all instances of the following applications:
   - Interactive SQL
   - Microsoft Excel
2. Delete the Excel workbook *order_central.xlsx*.
3. Close the SQL Anywhere, MobiLink, and synchronization client windows.
4. Delete all tutorial-related DSNs.
   a. Start the ODBC Administrator.

      Type the following command at the command prompt:

      ```
      odbcad32
      ```

   b. Remove the **excel_datasource** and **mlexcel_db** data sources.
5. Delete the consolidated and remote databases.
   a. Navigate to the directory containing your consolidated and remote databases.
   b. Delete *MLconsolidated.db*, *MLconsolidated.log*, *remote1.db*, and *remote1.log*.

Further reading

For more information about MobiLink synchronization, see "Understanding MobiLink synchronization" on page 1.

For more information about MobiLink clients, see "MobiLink clients" [*MobiLink - Client Administration*].

For more information about MobiLink direct row handling, see "Direct row handling" [*MobiLink - Server Administration*].

Tutorial: Synchronizing with XML

Introduction to synchronizing with XML

This tutorial shows you how to implement MobiLink direct row handling so that you can use a data source other than a supported consolidated data source. This tutorial uses a Java implementation as an example.

You can use direct row handling to communicate remote data to any central data source, application, or web service.

For more information about direct row handling, see "Direct row handling" [*MobiLink - Server Administration*].

This tutorial shows you how to synchronize data in an XML file to remote clients.

Required software

- SQL Anywhere 12
- Java Software Development Kit
- XML DOM library

Competencies and experience

You require:

- Familiarity with Java
- Familiarity with XML
- Familiarity with XML DOM
- Basic knowledge of MobiLink event scripts and MobiLink synchronization

Goals

You gain competence and familiarity with:

- The MobiLink server Java API for Java
- Creating methods for MobiLink direct row handling

Suggested background reading

- "Understanding MobiLink synchronization" on page 1
- "Synchronization techniques" [*MobiLink - Server Administration*]
- "Direct row handling" [*MobiLink - Server Administration*]

MobiLink code exchange samples are located at http://www.sybase.com/detail?id=1058600#319.

You can post questions on the MobiLink newsgroup at sybase.public.sqlanywhere.mobilink.

For more information about using XML DOM with Java, see http://java.sun.com/webservices/jaxp/.

Lesson 1: Set up an XML datasource

In this section, you create an XML file to store order information.

To set up an XML datasource

1. Create an XML file with the following contents:

```
<?xml version="1.0" encoding="UTF-8"?>
<orders></orders>
```


2. Save the XML file.

   This tutorial assumes *c:\MLobjxml* as the working directory for server-side components. Save the XML file as *order_comments.xml* in this directory.

Lesson 2: Set up your MobiLink consolidated database

Your MobiLink consolidated database is a central repository of data and includes MobiLink system tables and stored procedures used to manage the synchronization process. With direct row handling, you synchronize with a data source other than a consolidated database, but you still need a consolidated database to maintain information used by the MobiLink server.

In this lesson, you perform the following tasks:

- Create a database and define an ODBC data source.
- Add data tables to synchronize to remote clients.
- Install MobiLink system tables and stored procedures.

Note
If you already have a MobiLink consolidated database set up with MobiLink system objects and a DSN, you can skip this lesson.

Create your consolidated database

In this tutorial, you create a SQL Anywhere database using the Sybase Central **Create Database Wizard**.

To create your SQL Anywhere RDBMS

1. Choose **Start** » **Programs** » **SQL Anywhere 12** » **Administration Tools** » **Sybase Central**.
2. In Sybase Central, choose **Tools** » **SQL Anywhere 12** » **Create Database**.
3. Click **Next**.
4. Leave the default of **Create Database On This Computer**, and then click **Next**.
5. In the **Save The Main Database File To The Following File** field, type the file name and path for the database. For example, *c:\MLobjxml\MLconsolidated.db*. Click **Next**.
6. Follow the remaining instructions in the **Create Database Wizard** and accept the default values. On the **Connect To The Database** page, clear the **Stop The Database After Last Disconnect** option.
7. Click **Finish**.

   The MLconsolidated database appears in Sybase Central.

8. Click **Close** on the **Creating Database** window.

Define an ODBC data source for the consolidated database

Use the SQL Anywhere 12 driver to define an ODBC data source for the MLconsolidated database.

To define an ODBC data source for the consolidated database

1. From the Sybase Central **Tools** menu, choose **SQL Anywhere 12** » **Open ODBC Administrator**.
2. Click the **User DSN** tab, and click **Add**.
3. In the **Create New Data Source** window, click **SQL Anywhere 12** and click **Finish**.
4. Perform the following tasks in the **ODBC Configuration For SQL Anywhere** window:
   a. Click the **ODBC** tab.
   b. In the **Data Source Name** field, type **mlxml_db**.
   c. Click the **Login** tab.
   d. In the **User ID** field, type **DBA**.
   e. In the **Password** field, type **sql**.
   f. In the **Server Name** field, type **MLconsolidated**.
   g. Click **OK**.
5. Close ODBC data source administrator.

   Click **OK** on the **ODBC Data Source Administrator** window.

Create tables for synchronization

In this procedure, you create the RemoteOrders table in the MobiLink consolidated database. The RemoteOrders table contains the following columns:

Column	Description
order_id	A unique identifier for orders.
product_id	A unique identifier for products.
quantity	The number of items sold.
order_status	The order status.
last_modified	The last modification date of a row. You use this column for timestamp based downloads, a common technique used to filter rows for efficient synchronization.

To create the RemoteOrders table

1. Connect to your database using Interactive SQL.

   You can start Interactive SQL from Sybase Central or from a command prompt.

- To start Interactive SQL from Sybase Central, right-click the **MLconsolidated - DBA** database and click **Open Interactive SQL**.
- To start Interactive SQL at a command prompt, run the following command:

  ```
  dbisql -c "dsn=mlxml_db"
  ```

2. Run the following SQL script in Interactive SQL to create the RemoteOrders table:

   ```
   CREATE TABLE RemoteOrders (
       order_id           INTEGER NOT NULL,
       product_id         INTEGER NOT NULL,
       quantity           INTEGER,
       order_status       VARCHAR(10) DEFAULT 'new',
       last_modified      TIMESTAMP DEFAULT CURRENT TIMESTAMP,
       PRIMARY KEY(order_id)
   )
   ```

   Interactive SQL creates the RemoteOrders table in your consolidated database.

Stay connected in Interactive SQL for the next procedure.

Run MobiLink setup scripts

You can find setup scripts for each supported consolidated database in the *MobiLink/setup* subdirectory of your SQL Anywhere 12 installation.

In this procedure you set up a SQL Anywhere consolidated database. You do this using the *syncsa.sql* setup script. Running *syncsa.sql* creates a series of system tables and stored procedures prefaced with **ml_**. The MobiLink server works with these tables and stored procedures in the synchronization process.

To install MobiLink system tables

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "dsn=mlxml_db"
   ```

2. Run the following command in Interactive SQL to create MobiLink system tables and stored procedures. Replace *C:\Program Files\SQL Anywhere 12* with the location of your SQL Anywhere 12 installation.

   ```
   READ "C:\Program Files\SQL Anywhere 12\MobiLink\setup\syncsa.sql"
   ```

   Interactive SQL applies *syncsa.sql* to your consolidated database.

Stay connected in Interactive SQL for the next lesson.

Further reading

For information about creating SQL Anywhere databases, see "Initialization utility (dbinit)" [*SQL Anywhere Server - Database Administration*].

For information about creating tables, see "CREATE TABLE statement" [*SQL Anywhere Server - SQL Reference*].

For information about setting up MobiLink consolidated databases, see "MobiLink consolidated databases" [*MobiLink - Server Administration*].

Lesson 3: Add synchronization scripts

In this lesson you add scripts to your consolidated database for SQL row handling and direct row handling.

SQL row handling

SQL row handling allows you to synchronize remote data with tables in your MobiLink consolidated database. SQL-based scripts define:

- How data that is uploaded from a MobiLink client is to be applied to the consolidated database.
- What data should be downloaded from the consolidated database.

In this lesson, you write synchronization scripts for the following SQL-based upload and download events:

- **upload_insert** This event defines how new orders inserted in a client database should be applied to the consolidated database.
- **download_cursor** This event defines the orders that should be downloaded to remote clients.
- **download_delete_cursor** This event is required when using synchronization scripts that are not upload-only. You set the MobiLink server to ignore this event for the purpose of this tutorial.

In this procedure, you add synchronization script information to your MobiLink consolidated database using stored procedures.

To add SQL-based scripts to MobiLink system tables

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "dsn=mlxml_db"
   ```

2. Use the ml_add_table_script stored procedure to add SQL-based table scripts for the upload_insert, download_cursor and download_delete_cursor events.

   Run the following SQL script in Interactive SQL. The upload_insert script inserts the uploaded order_id, product_id, quantity, and order_status into the MobiLink consolidated database. The download_cursor script uses timestamp-based filtering to download updated rows to remote clients.

   ```
   CALL ml_add_table_script( 'default', 'RemoteOrders',
       'upload_insert',
       'INSERT INTO RemoteOrders( order_id, product_id, quantity,
   order_status)
       VALUES( ?, ?, ?, ? )' );

   CALL ml_add_table_script( 'default', 'RemoteOrders',
       'download_cursor',
       'SELECT order_id, product_id, quantity, order_status
   ```



```
        FROM RemoteOrders WHERE last_modified >= ?');

    CALL ml_add_table_script( 'default', 'RemoteOrders',
        'download_delete_cursor', '--{ml_ignore}');

    COMMIT
```

Direct row handling processing

In this tutorial, you use direct row handling to add special processing to a SQL-based synchronization system. In this procedure you register method names corresponding to the handle_UploadData, download_cursor, and download_delete_cursor events. You create your own Java class in "Lesson 4: Create a Java class for MobiLink direct row handling" on page 196.

To add information for direct row handling in MobiLink system tables

1. Connect to your consolidated database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "dsn=mlxml_db"
   ```

2. Register the Java method for the handle_UploadData event.

   Run the following SQL script in Interactive SQL:

   ```
   CALL ml_add_java_connection_script( 'default',
       'handle_UploadData', 'MobiLinkOrders.GetUpload' );
   ```

   Interactive SQL registers the GetUpload method for the handle_UploadData event. You create the GetUpload method, which retrieves inserted data from the OrderComments table in the MobiLink client database, in an upcoming lesson.

3. Register the download_cursor and download_delete_cursor events.

   Run the following SQL script in Interactive SQL:

   ```
   CALL ml_add_table_script( 'default', 'OrderComments',
       'download_cursor', '--{ml_ignore}');

   CALL ml_add_table_script( 'default', 'OrderComments',
       'download_delete_cursor', '--{ml_ignore}');
   ```

   The download_cursor and download_delete_cursor events must be registered for the OrderComments table when using scripts because the synchronization is bi-directional and not upload-only. See "Required scripts" [*MobiLink - Server Administration*].

4. Commit your changes.

   Run the following SQL script in Interactive SQL:

   ```
   COMMIT;
   ```

You can now close Interactive SQL.

Further reading

For information about using SQL-based events to upload data from remote clients to a MobiLink consolidated database, see:

- "Writing scripts to upload rows" [*MobiLink - Server Administration*]
- "upload_insert table event" [*MobiLink - Server Administration*]
- "upload_update table event" [*MobiLink - Server Administration*]
- "upload_delete table event" [*MobiLink - Server Administration*]

For information about uploading data to data sources other than consolidated databases, see "Handling direct uploads" [*MobiLink - Server Administration*].

For information about using SQL-based events to download data from a MobiLink consolidated database, see:

- "Writing scripts to download rows" [*MobiLink - Server Administration*]
- "download_cursor table event" [*MobiLink - Server Administration*]
- "download_delete_cursor table event" [*MobiLink - Server Administration*]

For information about downloading data to data sources other than consolidated databases, see "Handling direct downloads" [*MobiLink - Server Administration*].

For information about the synchronization event sequence, see "Overview of MobiLink events" [*MobiLink - Server Administration*].

For information about synchronization techniques for download filtering, see "Timestamp-based downloads" [*MobiLink - Server Administration*] and "Partitioning rows among remote databases" [*MobiLink - Server Administration*].

For information about managing scripts, see "Adding and deleting scripts" [*MobiLink - Server Administration*].

For information about direct row handling, see "Direct row handling" [*MobiLink - Server Administration*].

Lesson 4: Create a Java class for MobiLink direct row handling

In this lesson, you use direct row handling to process rows in the OrderComments table in your client database. You add the following methods for direct row handling:

- **GetUpload** You use this method for the handle_UploadData event. GetUpload writes uploaded comments to the XML file.

The following procedure shows you how to create a Java class including your methods for processing. For a complete listing, see "Complete MobiLinkOrders Java code listing" on page 201.

To create a Java class for download-only direct row handling

1. Create a class named MobiLinkOrders.

   Write the following code:

   ```
   import ianywhere.ml.script.*;
   import java.io.*;
   import java.sql.*;

   import javax.xml.parsers.DocumentBuilder;
   import javax.xml.parsers.DocumentBuilderFactory;
   import javax.xml.parsers.ParserConfigurationException;

   import org.xml.sax.SAXException;

   import org.w3c.dom.Document;
   import org.w3c.dom.Element;
   import org.w3c.dom.Node;
   import org.w3c.dom.NodeList;

   // For write operation
   import javax.xml.transform.Transformer;
   import javax.xml.transform.TransformerException;
   import javax.xml.transform.TransformerFactory;
   import javax.xml.transform.TransformerConfigurationException;
   import javax.xml.transform.dom.DOMSource;
   import javax.xml.transform.stream.StreamResult;

   public class MobiLinkOrders {
   ```

2. Declare a class-level DBConnectionContext instance and Document instance. Document is a class that represents an XML document as an object.

   Write the following code:

   ```
       // Class level DBConnectionContext
       DBConnectionContext _cc;
       Document _doc;
   ```

   The MobiLink server passes a DBConnectionContext instance to your class constructor. DBConnectionContext encapsulates information about the current connection with the MobiLink consolidated database.

3. Create your class constructor.

   Your class constructor sets your class-level DBConnectionContext instance.

   Write the following code:

   ```
       public MobiLinkOrders( DBConnectionContext cc ) throws IOException,
   FileNotFoundException {
           // Declare a class-level DBConnectionContext
           _cc = cc;
       }
   ```

4. Write the GetUpload method.

The GetUpload method obtains an UploadedTableData class instance representing the OrderComments table. The OrderComments table contains special comments made by remote sales employees. You create this table in "Lesson 6: Set up your MobiLink client database" on page 204. The UploadedTableData getInserts method returns a result set for new order comments.

a. Write the method declaration.

   Write the following code:

```
    // Method for the handle_UploadData synchronization event
    public void GetUpload( UploadData ut ) throws SQLException,
IOException {
```

b. Write code that retrieves any uploaded inserts from the MobiLink client.

   Write the following code:

```
        // Get an UploadedTableData for the remote table
        UploadedTableData remoteOrdersTable =
ut.getUploadedTableByName("OrderComments");

        // Get inserts uploaded by the MobiLink client
        // as a java.sql.ResultSet
        ResultSet insertResultSet = remoteOrdersTable.getInserts();
```

c. Write code that reads the existing XML file, **order_comments.xml**.

   Write the following code:

```
        readDom("order_comments.xml");
```

d. Write code that adds all uploaded inserts to the XML file.

   Write the following code:

```
        // Write out each insert in the XML file
        while( insertResultSet.next() ) {
            buildXML(insertResultSet);
        }
```

e. Write code that outputs to the XML file.

   Write the following code:

```
        writeXML();
```

f. Write code that closes the ResultSet.

   Write the following code:

```
        // Close the result set of uploaded inserts
        insertResultSet.close();
    }
```

5. Write the buildXML method.

   Write the following code:

```
    private void buildXML( ResultSet rs ) throws SQLException {
        int order_id = rs.getInt(1);
        int comment_id = rs.getInt(2);
        String order_comment = rs.getString(3);

        // Create the comment object to be added to the XML file
```



```
            Element comment = _doc.createElement("comment");
            comment.setAttribute("id", Integer.toString(comment_id));
            comment.appendChild(_doc.createTextNode(order_comment));

            // Get the root element (orders)
            Element root = _doc.getDocumentElement();

            // Get each individual order
            NodeList rootChildren = root.getChildNodes();
            for(int i = 0; i < rootChildren.getLength(); i++) {
                // If the order exists, add the comment to the order
                Node n = rootChildren.item(i);
                if(n.getNodeType() == Node.ELEMENT_NODE) {
                    Element e = (Element) n;
                    int idIntVal = Integer.parseInt(e.getAttribute("id"));
                    if(idIntVal == order_id) {
                        e.appendChild(comment);
                       // The comment has been added to the file, so exit the
function
                        return;
                    }
                }
            }

            // If the order did not exist already, create it
            Element order = _doc.createElement("order");
            order.setAttribute("id", Integer.toString(order_id));
            // Add the comment to the new order
            order.appendChild(comment);
            root.appendChild(order);
        }
```

6. Write the writeXML method.

   Write the following code:

```
        private void writeXML() {
            try {
                // Use a Transformer for output
                TransformerFactory tFactory =
TransformerFactory.newInstance();
                Transformer transformer = tFactory.newTransformer();

                // The XML source is _doc
                DOMSource source = new DOMSource(_doc);
                // Write the xml data to order_comments.xml
                StreamResult result = new StreamResult(new
File("order_comments.xml"));
                transformer.transform(source, result);
            } catch (TransformerConfigurationException tce) {
                // Error generated by the parser
                System.out.println ("\n** Transformer Factory error");
                System.out.println("   " + tce.getMessage() );

                // Use the contained exception, if any
                Throwable x = tce;
                if (tce.getException() != null) x = tce.getException();
                x.printStackTrace();

            } catch (TransformerException te) {
                // Error generated by the parser
                System.out.println ("\n** Transformation error");
                System.out.println("   " + te.getMessage() );
```

```
                // Use the contained exception, if any
                Throwable x = te;
                if (te.getException() != null) x = te.getException();
                x.printStackTrace();
            }
        }
```

7. Write the readDOM method.

   Write the following code:

```
    private void readDom(String filename) {
        DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();

        try {
            // Parse the Document data into _doc
            DocumentBuilder builder = factory.newDocumentBuilder();
            _doc = builder.parse( new File(filename) );

        } catch (SAXException sxe) {
            // Error generated during parsing)
            Exception x = sxe;
            if (sxe.getException() != null) x = sxe.getException();
            x.printStackTrace();

        } catch (ParserConfigurationException pce) {
            // Parser with specified options can't be built
            pce.printStackTrace();

        } catch (IOException ioe) {
            // I/O error
            ioe.printStackTrace();
        }
    }
}
```

8. Save your Java code as *MobiLinkOrders.java* in your working directory *c:\MLobjxml*.

   See "Complete MobiLinkOrders Java code listing" on page 201 to verify the code in *MobiLinkOrders.java*.

9. Compile your class file.

   a. Navigate to the directory containing your Java source files.

   b. Compile MobiLinkOrders that refer to the MobiLink server API library for Java.

      You need to reference *mlscript.jar* located in *install-dir\Java* and make sure that you have the XML DOM library installed correctly.

      Run the following command, replacing *C:\Program Files\SQL Anywhere 12* with your SQL Anywhere 12 directory:

```
javac -classpath "C:\Program Files\SQL Anywhere 12\java\mlscript.jar"
MobiLinkOrders.java
```

Further reading

For more information about synchronization logic, see "Writing synchronization scripts in Java" [*MobiLink - Server Administration*].

For more information about direct row handling, see "Direct row handling" [*MobiLink - Server Administration*].

Complete MobiLinkOrders Java code listing

The following listing shows the complete Java MobiLinkOrders class code used for this tutorial. For a step by step explanation, see "Lesson 4: Create a Java class for MobiLink direct row handling" on page 196.

```
import ianywhere.ml.script.*;
import java.io.*;
import java.sql.*;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;

import org.xml.sax.SAXException;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

// For write operation
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.TransformerConfigurationException;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

public class MobiLinkOrders {

    // Class level DBConnectionContext
    DBConnectionContext _cc;
    Document _doc;

    public MobiLinkOrders( DBConnectionContext cc ) throws IOException,
FileNotFoundException {
        // Declare a class-level DBConnectionContext
        _cc = cc;
    }

    //  Method for the handle_UploadData synchronization event
    public void GetUpload( UploadData ut ) throws SQLException, IOException {
        // Get an UploadedTableData for the remote table
        UploadedTableData remoteOrdersTable =
ut.getUploadedTableByName("OrderComments");

        // Get inserts uploaded by the MobiLink client
        // as a java.sql.ResultSet
        ResultSet insertResultSet = remoteOrdersTable.getInserts();

        readDom("order_comments.xml");

        // Write out each insert in the XML file
        while( insertResultSet.next() ) {
            buildXML(insertResultSet);
        }
```

```
            writeXML();

            // Close the result set of uploaded inserts
            insertResultSet.close();
        }

        private void buildXML( ResultSet rs ) throws SQLException {
            int order_id = rs.getInt(1);
            int comment_id = rs.getInt(2);
            String order_comment = rs.getString(3);

            // Create the comment object to be added to the XML file
            Element comment = _doc.createElement("comment");
            comment.setAttribute("id", Integer.toString(comment_id));
            comment.appendChild(_doc.createTextNode(order_comment));

            // Get the root element (orders)
            Element root = _doc.getDocumentElement();

            // Get each individual order
            NodeList rootChildren = root.getChildNodes();

            for(int i = 0; i < rootChildren.getLength(); i++) {
                // If the order exists, add the comment to the order
                Node n = rootChildren.item(i);
                if(n.getNodeType() == Node.ELEMENT_NODE) {
                    Element e = (Element) n;
                    int idIntVal = Integer.parseInt(e.getAttribute("id"));

                    if(idIntVal == order_id) {
                        e.appendChild(comment);
                        // The comment has been added to the file, so exit the
    function
                        return;
                    }
                }
            }

            // If the order did not exist already, create it
            Element order = _doc.createElement("order");
            order.setAttribute("id", Integer.toString(order_id));

            // Add the comment to the new order
            order.appendChild(comment);
            root.appendChild(order);
        }

        private void writeXML() {
            try {
                // Use a Transformer for output
                TransformerFactory tFactory = TransformerFactory.newInstance();
                Transformer transformer = tFactory.newTransformer();

                // The XML source is _doc
                DOMSource source = new DOMSource(_doc);
                // Write the xml data to order_comments.xml
                StreamResult result = new StreamResult(new
    File("order_comments.xml"));
                transformer.transform(source, result);
            } catch (TransformerConfigurationException tce) {
                // Error generated by the parser
                System.out.println ("\n** Transformer Factory error");
                System.out.println("   " + tce.getMessage() );
```



```
            // Use the contained exception, if any
            Throwable x = tce;
            if (tce.getException() != null) x = tce.getException();
            x.printStackTrace();
        } catch (TransformerException te) {
            // Error generated by the parser
            System.out.println ("\n** Transformation error");
            System.out.println("   " + te.getMessage() );

            // Use the contained exception, if any
            Throwable x = te;
            if (te.getException() != null) x = te.getException();
            x.printStackTrace();
        }
    }

    private void readDom(String filename) {
        DocumentBuilderFactory factory =
DocumentBuilderFactory.newInstance();

        try {
            //parse the Document data into _doc
            DocumentBuilder builder = factory.newDocumentBuilder();
            _doc = builder.parse( new File(filename) );

        } catch (SAXException sxe) {
            // Error generated during parsing)
            Exception x = sxe;
            if (sxe.getException() != null) x = sxe.getException();
            x.printStackTrace();

        } catch (ParserConfigurationException pce) {
            // Parser with specified options can't be built
            pce.printStackTrace();

        } catch (IOException ioe) {
            // I/O error
            ioe.printStackTrace();
        }
    }
}
```

Lesson 5: Start the MobiLink server

In this lesson, you start the MobiLink server. You start the MobiLink server (mlsrv12) using the -c option to connect to your consolidated database, and the -sl java option to load your Java class.

To start the MobiLink server (mlsrv12)

- Connect to your consolidated database and load the class on the mlsrv12 command line.

  Replace *c:\MLobjxml* with the location of your source files.

  Run the following command:

  ```
  mlsrv12 -c "dsn=mlxml_db" -o serverOut.txt -v+ -dl -zu+ -x tcpip  -sl java
  (-cp c:\MLobjxml)
  ```

The MobiLink server messages window appears.

Below is a description of each MobiLink server option used in this tutorial. The options -o, -v, and -dl provide debugging and troubleshooting information. Using these logging options is appropriate in a development environment. For performance reasons, -v and -dl are typically not used in production.

Option	Description
-c	Precedes the connection string.
-o	Specifies the message log file *serverOut.txt*.
-v+	The -v option specifies what information is logged. Using -v+ sets maximum verbose logging.
-dl	Displays all log messages on screen.
-zu+	Adds new users automatically.
-x	Sets the communications protocol and parameters for MobiLink clients.
-sl java	Specifies a set of directories to search for class files, and forces the Java VM to load on server startup.

Further reading

For a complete list of MobiLink server options, see "MobiLink server options" [*MobiLink - Server Administration*].

For more information about loading Java classes, see "-sl java mlsrv12 option" [*MobiLink - Server Administration*].

Lesson 6: Set up your MobiLink client database

In this lesson, you use a SQL Anywhere database for your consolidated database and your MobiLink client. For tutorial purposes your MobiLink client, consolidated database, and MobiLink server all reside on the same computer.

To set up the MobiLink client database, you create the RemoteOrders and OrderComments tables. The RemoteOrders table corresponds to the RemoteOrders table on the consolidated database. The MobiLink server uses SQL-based scripts to synchronize remote orders. The OrderComments table is only used on client databases. The MobiLink server processes the OrderComments tables using special events.

You also create a synchronization user, publication, and subscription on the client database.

To set up your MobiLink client database

1. Create your MobiLink client database using the dbinit command line utility.

   Run the following command:


```
dbinit -i -k remote1
```

The -i and -k options omit jConnect support and Watcom SQL compatibility views, respectively.

2. Start your MobiLink client database using the dbeng12 command line utility.

   Run the following command:

   ```
   dbeng12 remote1
   ```

3. Connect to your MobiLink client database using Interactive SQL.

   Run the following command:

   ```
   dbisql -c "server=remote1;uid=DBA;pwd=sql"
   ```

4. Create the RemoteOrders table.

   Run the following SQL script in Interactive SQL:

   ```
   CREATE TABLE RemoteOrders (
       order_id           INTEGER NOT NULL,
       product_id         INTEGER NOT NULL,
       quantity           INTEGER,
       order_status       VARCHAR(10) DEFAULT 'new',
       PRIMARY KEY(order_id)
   )
   ```

5. Create the OrderComments table.

   Run the following SQL script in Interactive SQL:

   ```
   CREATE TABLE OrderComments (
       comment_id         INTEGER NOT NULL,
       order_id           INTEGER NOT NULL,
       order_comment      VARCHAR(255),
       PRIMARY KEY(comment_id),
       FOREIGN KEY(order_id) REFERENCES RemoteOrders(order_id)
   )
   ```

6. Create your MobiLink synchronization user, publication, and subscription.

   Run the following SQL script in Interactive SQL:

   ```
   CREATE SYNCHRONIZATION USER ml_sales1;
   CREATE PUBLICATION order_publ (TABLE RemoteOrders, TABLE OrderComments);
   CREATE SYNCHRONIZATION SUBSCRIPTION TO order_publ FOR ml_sales1
    TYPE TCPIP ADDRESS 'host=localhost'
   ```

   Note
   You specify how to connect to the MobiLink server using the TYPE and ADDRESS clauses in the CREATE SYNCHRONIZATION SUBSCRIPTION statement.

   You can use publications to determine what data is synchronized. In this case you specify the entire RemoteOrders and OrderComments tables.

Stay connected in Interactive SQL for the next lesson.

Further reading

For information about creating a SQL Anywhere database, see "Initialization utility (dbinit)" [*SQL Anywhere Server - Database Administration*].

For information about MobiLink clients, see "MobiLink clients" [*MobiLink - Client Administration*].

For information about creating MobiLink objects on the client, see:

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

Lesson 7: Synchronize

The dbmlsync utility initiates MobiLink synchronization for SQL Anywhere remote databases. Before starting dbmlsync, add order data and comments to your remote database.

To set up your remote data (client-side)

1. Connect to the MobiLink client database in Interactive SQL if you are not already connected.

   Run the following command:

   ```
   dbisql -c "server=remote1;uid=DBA;pwd=sql"
   ```

2. Add an order to the RemoteOrders table in the client database.

   Run the following SQL script in Interactive SQL:

   ```
   INSERT INTO RemoteOrders (order_id, product_id, quantity, order_status)
    VALUES (1,12312,10,'new')
   ```

3. Add a comment to the OrderComments table in the client database.

   Run the following SQL script in Interactive SQL:

   ```
   INSERT INTO OrderComments (comment_id, order_id, order_comment)
    VALUES (1,1,'send promotional material with the order')
   ```

4. Commit your changes.

   Run the following SQL script in Interactive SQL:

   ```
   COMMIT;
   ```

To start the synchronization client (client-side)

- Run the following command at the command prompt:


```
dbmlsync -c "server=remote1;uid=DBA;pwd=sql" -e scn=on -o rem1.txt -v+
```

The following table contains a description for each dbmlsync option used in this lesson:

Option	Description
-c	Specifies the connection string.
-e scn	Sets SendColumnNames to on. This is required by direct row handling if you want to reference columns by name.
-o	Specifies the message log file *rem1.txt*.
-v+	The -v option specifies what information is logged. Using -v+ sets maximum verbose logging.

Once you have started the MobiLink synchronization client, an output screen appears indicating that the synchronization succeeded. SQL-based synchronization transferred rows in the client RemoteOrders table to the RemoteOrders table in the consolidated database.

Java processing inserted your comment in the XML file.

Further reading

For more information about dbmlsync, see "SQL Anywhere clients" [*MobiLink - Client Administration*].

Cleanup

Remove tutorial materials from your computer.

To remove tutorial materials from your computer

1. Close all instances of Interactive SQL.
2. Close the SQL Anywhere, MobiLink, and synchronization client windows.
3. Delete all tutorial-related DSNs.
   a. Start the ODBC Administrator.

      Type the following command at the command prompt:

      ```
      odbcad32
      ```

   b. Remove the **mlxml_db** data source.
4. Delete the consolidated and remote databases.
   a. Navigate to the directory containing your consolidated and remote databases.
   b. Delete *MLconsolidated.db*, *MLconsolidated.log*, *remote1.db*, and *remote1.log*.

Further reading

For more information about MobiLink synchronization, see "Understanding MobiLink synchronization" on page 1.

For more information about MobiLink clients, see "MobiLink clients" [*MobiLink - Client Administration*].

For more information about MobiLink direct row handling, see "Direct row handling" [*MobiLink - Server Administration*].

Tutorial: Using central administration of remote databases

Introduction to central administration of remote databases tutorial

This tutorial leads you through the process of setting up central administration of remote databases and demonstrates how several common operations can be performed.

You may follow this tutorial to either set up central administration from scratch or to add central administration to an existing synchronization system. Throughout the procedure, the tutorial will point out where you should do different things if you are adding central administration to an existing synchronization system.

Required software

This tutorial assumes you have a complete install of SQL Anywhere, including MobiLink and Sybase Central on your local computer where you are running the tutorial.

For information about deploying the MobiLink Agent, see "Deploying SQL Anywhere MobiLink clients" [*MobiLink - Server Administration*] and "Deploying UltraLite MobiLink clients" [*MobiLink - Server Administration*].

Overview

This tutorial shows you how to:

- "Lesson 1: Create a consolidated database" on page 209
- "Lesson 2: Create a MobiLink project" on page 210
- "Lesson 3: Start the MobiLink server" on page 210
- "Lesson 4: Define a remote schema name" on page 211
- "Lesson 5: Define a MobiLink user" on page 211

Suggested background reading

For an overview of central administration of remote databases, see "Central administration of remote databases" [*MobiLink - Server Administration*].

Lesson 1: Create a consolidated database

In this lesson, you set up a consolidated database. If you have an existing synchronization system you can skip this section.

To create a consolidated database

1. Run the following commands to create directories to be used in this tutorial. The consolidated directory contains all the database and other files that would normally reside on your central server.

   ```
   md c:\cadmin_demo
   md c:\cadmin_demo\consolidated
   ```

2. Create a SQL Anywhere consolidated database and an ODBC data source to connect to it.

   ```
   cd c:\cadmin_demo\consolidated
   dbinit consol.db
   ```

```
start dbeng12 consol.db
dbdsn -w cadmin_tutorial_consol consol -y -c
"uid=dba;pwd=sql;dbf=consol.db;server=consol"
cd ..
```

Lesson 2: Create a MobiLink project

To perform central administration, you must create a MobiLink project. The project acts as a container for the various objects you define for central administration.

To create a MobiLink project

1. Start Sybase Central as follows:

   ```
   scjview
   ```

2. In the **Folders** view in the left pane, right-click **MobiLink 12** and choose **New** » **Project**. The **Create Project Wizard** appears.

3. On the **Welcome** page, change the project name to **Central Admin Tutorial** and accept the default location for the Project file. Click **Next**.

4. On the **Specify a Consolidated Database** page, check **Add a Consolidated Database to the Project**. Type **Tutorial** for the **Database Display Name**.

5. If you have an existing synchronization system, enter the connection string for your consolidated database in the **Connection String** field. Otherwise, enter the following for the **Connection String**:

   ```
   uid=dba;pwd=sql;dsn=cadmin_tutorial_consol
   ```

6. Check **Remember The Password** and click **Finish** to complete the wizard.

7. If this is the first time the consolidated database has been used by MobiLink, a message appears asking if you want to install the MobiLink system setup. Installing the MobiLink system setup adds MobiLink system tables and procedures. Click **Yes** then **OK**.

Lesson 3: Start the MobiLink server

The MobiLink server is needed both to synchronize data from your remote database and to synchronize tasks and task results between the consolidated database and the agent database on each remote device. Use the procedure in this lesson to start the MobiLink server.

If you have an existing synchronization system you can skip this lesson since you already have the server running. However, you should check your server command line and ensure that the -ftr and -ftru options are specified. These options are required to download files to your remote devices and to upload files from your remote devices.

To start the MobiLink server

- Execute the following in a command prompt:

```
md c:\cadmin_demo\consolidated\upload
md c:\cadmin_demo\consolidated\download
cd c:\cadmin_demo\consolidated
start mlsrv12.exe -c "dsn=cadmin_tutorial_consol;uid=dba;pwd=sql" -ftr
download -ftru upload -x tcpip(port=2439) -v+ -ot mlsrv.txt
cd ..
```

  This command starts the MobiLink server and creates the upload and download directories that contain files to be uploaded from or downloaded to remote devices. Following is a summary of the options used:

  - **-c** Specifies the connection parameters MobiLink uses to connect to the consolidated database.
  - **-ftr** Specifies the directory where MobiLink looks for files to download.
  - **-ftru** Specifies the directory where MobiLink puts files that are uploaded.
  - **-x** Specifies communication parameters that define how synchronization clients may connect to the MobiLink server.
  - **-v+** Specifies maximum verbosity. This is helpful for debugging but can slow performance in a production environment.
  - **-ot** Specifies the file where MobiLink output messages are logged.

Lesson 4: Define a remote schema name

The next step is to define a remote schema name. A remote schema name identifies a group of remote application databases that all share the same schema. Typically these would be databases being used by the same version of a particular application.

To define a remote schema name

1. Return to the **Folders** view in Sybase Central. Under **Central Admin Tutorial**, right-click **Remote Schema Names** and choose **New** » **Remote Schema Name**. The **Create Remote Schema Name Wizard** appears.
2. Type **Tutorial Application v1.0** for the schema name.
3. Select **SQL Anywhere** as the database type and click **Finish**.

Lesson 5: Define a MobiLink user

When an Agent synchronizes its agent database it must authenticate itself to the MobiLink server. It does this by using a MobiLink user and optionally a password. Normally you would use the same MobiLink user and password to synchronize your remote databases that the Agent uses to synchronize the agent database.

In this lesson, you define a MobiLink user for the Agent to use. You can skip this lesson if you have an existing synchronization system, and you want the MobiLink Agent to use one of your existing MobiLink users to synchronize.

To define a MobiLink user

1. In the **Folders** view under **Central Admin Tutorial** » **Consolidated Databases** » **Tutorial**, right-click **Users** and choose **New** » **User**. The **Create User Wizard** appears.

2. On the **Welcome** page, type **JOHN** for the name of the new user and click **Next**.

3. On the **Specify a Password** page, check **This User Will Require a Password to Connect** and type **sql** in both the **Password** and **Confirm Password** fields. Click **Finish**.

   If you do not want to authenticate Agents that try to synchronize, skip this step and add the -zu+ option to the MobiLink command line. When -zu+ is specified, each MobiLink user is registered when it first attempts to synchronize. See "-zu mlsrv12 option" [*MobiLink - Server Administration*].

Lesson 6: Define an Agent

Next you define an Agent. This Agent represents an instance of the MobiLink Agent running on a remote device. You must create a separate Agent for each remote device you are managing.

To define an Agent

1. In the **Folders** view, under **Central Admin Tutorial** » **Consolidated Databases** » **Tutorial**, right-click **Agents** and choose **New** » **Agent**. The **Create MobiLink Agent Wizard** appears.

2. On the **Welcome** page, choose **Set Up a Single Agent** and click **Next**.

3. On the **Agent ID** page, type **AID_JOHN** for the **Agent ID**. The Agent ID can be any value you like but each Agent must have a unique ID. By convention, Agent IDs begin with the prefix AID_ and usually the second part of the agent ID is the MobiLink user name used by the Agent. Click **Next**.

4. The **Remote Database** page lets you define a remote database to be managed by this Agent. This does not actually create the database; you will do that later. For **Remote Schema Name**, select **Tutorial Application v1.0**, which is the name you defined in the previous lesson, from the dropdown list.

5. If you have an existing synchronization system, fill in the **Connection String** field with a connection string that the MobiLink Agent can use to connect to the remote database that is already on your device.

   If this is a new synchronization system, enter the following in the **Database Connection String** field:

   ```
   start=dbeng12;eng=tutorial_v1;dbf={db_location}
   \tutorial_v1.db;uid=dba;pwd=sql
   ```

   Notice that this value uses the macro {db_location}. This macro is replaced by the directory on the remote device where application databases are stored. Click **Next**.

6. On the **Agent Configuration** page, type **30** and choose **Seconds** for the **Synchronization Interval**. The synchronization interval controls how frequently the Agent synchronizes its agent database. Synchronizing the agent database is how an Agent receives new tasks to perform and uploads the results of tasks it has already performed.

7. On the **Agent Configuration** page, type **10** and choose **Seconds** for the **Administration Polling Interval**. The administration polling interval determines how frequently the Agent checks for requests from the server for it to synchronize or perform other actions.

8. Click **Finish**.

Note
The short values chosen for the synchronization interval and administration polling interval provide a very responsive Agent, which is important for a demonstration or for troubleshooting. However, using short values globally in a production system results in increased load on your server and reduced performance.

Lesson 7: Configure the Agent on the remote device

In this lesson, you run the MobiLink Agent. The MobiLink agent must be running on each remote device that is centrally administered. For this tutorial, the Agent runs on the same machine where the server is running.

To configure the agent on the remote device

1. Start by creating a directory that contains the files that would normally be on the remote device.

```
md c:\cadmin_demo\remote
cd c:\cadmin_demo\remote
```

2. Next, run the MobiLink Agent in configuration mode as follows:

```
mlagent -cr -db . -x tcpip{host=localhost;port=2439} -a AID_JOHN  -u JOHN
-p sql
```

This step creates an agent database and stores some configuration information in it. Once the specified options are stored in the database, the Agent shuts down. Following is a summary of the options you used:

- **-cr** Specifies that the Agent should run in configuration mode and that it should discard any settings stored during previous runs in configuration mode.
- **-db** Specifies where the Agent should create application databases. This becomes the value of the {db_location} macro.
- **-x** Specifies how the Agent should connect to the MobiLink server to synchronize its agent database (to receive new tasks and upload results of tasks it has run). If you are adding central administration to an existing synchronization system, you need to change the value specified for this option to an appropriate string for connecting to your MobiLink server.

- **-a** Specifies the Agent ID for this Agent. Notice that you specified the same Agent ID that you previously created in the consolidated database using Sybase Central.
- **-u** Specifies the MobiLink user the Agent uses when synchronizing the agent database. This value is used by the server primarily to authenticate the Agent.
- **-p** Specifies the password that goes with the MobiLink user specified with the -u option.

3. Lastly, you need to run the MobiLink Agent on the remote device. For this tutorial, you explicitly start the Agent running as follows:

   ```
   start mlagent -v9 -ot agent.txt
   ```

   Following is a summary of the options used to run the Agent in this lesson:

   - **-v9** Use maximum verbosity. This is great for a tutorial but in a production environment you might want to choose lower verbosity except when you are troubleshooting.
   - **-ot** Specifies the file where the Agent logs its output.

4. You should now have the MobiLink Agent running and it should be synchronizing successfully. To check, return to Sybase Central. In the **Folders** view, under **Central Admin Tutorial** » **Consolidated Databases** » **Tutorial** » **Agents** » click **AID_JOHN** and look at the **Events** tab in the right pane. You should see an entry that indicates the Agent's first synchronization.

Production considerations for Agent configuration

Keep the following in mind when using central administration in a production environment:

- You may need to change the values specified for the -u and -p options to an appropriate MobiLink user and password combination for your synchronization system.
- You might want to use the -on option to limit the size of the log file produced by the Agent.
- A remote device can only be remotely administered while the MobiLink Agent is running on it. You would likely want to take steps to ensure that the Agent is always running. Some strategies for this might include running the Agent as a service or adding the agent to the Run startup group in the registry.

Lesson 8: Create a synchronization model

In this lesson, you create a synchronization model. If you are adding central administration to existing synchronization system you can skip this lesson.

To create a synchronization model

1. You first need to define the tables for the remote database in the consolidated database. In the **Folders** view of Sybase Central under **Central Admin Tutorial** » **Consolidated Databases** » , right-click **Tutorial** and choose **Open Interactive SQL**.

2. In the **SQL Statements** pane type the following:

   ```
   create table customer(
       cust_id       integer primary key,
   ```



```
    f_name        varchar(100),
    l_name        varchar(100)
)
```

3. Press F5 to execute the SQL. Close Interactive SQL. You do not need to save your SQL statements.
4. In the **Folders** view of Sybase Central, right-click **Central Admin Tutorial** and choose **New** » **Synchronization Model**.
5. On the **Welcome** page, type **tutorial1** for the name of the new synchronization model.
6. On the **Primary Key Requirements** page check all three checkboxes to confirm that your schema meets the requirements for synchronization. Click **Next**.
7. On the **Consolidated Database Schema** page, choose the **Tutorial** database and click **Next**.
8. On the **Remote Database Schema** page, select **No, Create a New Remote Database Schema** and click **Next**.
9. On the **New Remote Database Schema** page, ensure the customer table is checked and click **Next**.
10. On the **Download Type** page, choose **Snapshot Download** and click **Next**.
11. On the **Download Deletes** page, answer **No** to the question **Do You Want Data Deleted on the Consolidated Database to be Deleted on the Remote Databases** and click **Next**.
12. On the **Download Subset** page, check **Yes, Download the Same Data to Each Remote** and click **Next**.
13. On the **Upload Conflict Detection** page, select **No Conflict Detection** and click **Next**.
14. Accept the defaults on the **Publication, Script Version and Description** page and click **Finish**.

    You have now created a synchronization model that contains a single table called **customer** that will be synchronized between the remote and the consolidated databases. The next step is to deploy that model to create synchronization objects in the consolidated database and to generate SQL for creating a remote database.

Lesson 9: Deploy the synchronization model

In this lesson, you will deploy the synchronization model you created in the previous lesson.

To deploy the synchronization model

1. In the **Folders** view of Sybase Central, under **Central Admin Tutorial** » **Synchronization Models**, right-click **tutorial1** and choose **Deploy**. The **Deploy Synchronization Model Wizard** appears.

2. On the **Welcome** page, select **Specify the Deployment Details For One or More of the Following** and check the **Consolidated Database** and **Remote Database and Synchronization Client** options and click **Next**.

3. On the **Consolidated Database Deployment Destination** page, uncheck **Save Changes to the Following SQL File** and check **Connect to the Consolidated Database to Directly Apply the Changes**. Choose the **Tutorial** database and click **Next**.

4. On the **Remote Database Deployment** page, choose **New SQL Anywhere Database** and click **Next**.

5. On the **New SQL Anywhere Remote Database** page, check **Make a Command File and a SQL file With Commands to Create a Database** and type **c:\cadmin_demo\remote_db\tutorial_v1.sql** in the **SQL file** field. Uncheck **Create a remote SQL Anywhere database**. Click **Next** to continue and **Yes** to confirm that you want the new directory and file to be created.

6. On the **MobiLink User and Subscription** page, type **{ml_username}** for the user name and **{ml_password}** for the password. These macro values are used in the generated SQL files and will be replaced with the MobiLink user and password being used by the MobiLink Agent when the SQL is executed on the remote device.

7. Check **Register This User in the Consolidated Database for MobiLink Authentication** and type **tutorial1** for the subscription name. Click **Next**.

8. On the **Synchronization Stream Parameters** page, choose **TCP/IP** and enter **2439** for the **Port**. These are the stream parameters that you used when you started the MobiLink server earlier. Click **Next** until you get to the **Advanced Options for SQL Anywhere Remote Synchronization Client** then click **Finish** and then **Yes** to create the directory.

   When you navigate away from the synchronization model, you are asked if you want to save your changes. Click **Yes**.

You have now completed creating and deploying a synchronization model. When you deployed the model, scripts were added to the consolidated database to allow a remote database to synchronize. You also generated a SQL file in the *cadmin_demo\remote_db* directory which can be used to create a remote database. You may like to look at those files now.

Lesson 10: Use a remote task

Most actions in central administration involve a remote task. A remote task is a collection of commands that is created by an administrator. It can be assigned to one or more Agents. Once assigned to an Agent, the remote task is downloaded to the Agent the next time the Agent synchronizes its agent database. The Agent then executes the task at an appropriate time and uploads information about the execution.

In this lesson, you use a remote task to display the message "Hello World" on the remote device. This is a very important section because you will depend heavily on the skills you develop here in the remaining sections of the tutorial.

To use a remote task

1. The first step is to create a new remote task. In the **Folders** view of Sybase Central under **Central Admin Tutorial**, right-click **Remote Tasks** and choose **New** » **Remote Task**. The **Create Remote Task Wizard** appears.

2. On the **Welcome** page, type **Hello World** in the **Name** field. Click **Next**.

3. On the **Trigger Mechanisms** page, check **When it is Received by an Agent** then click **Finish** to complete the wizard.

4. Click on the newly created **Hello World** task in the **Folders** view. In the right pane you see the **Commands** tab which allows you to add commands to your task.

5. On the **Commands** tab, select **Prompt** from the **Command Type** dropdown list. In the **Message** field, type **Hello World**.

6. To add a second command to the task, either press Tab twice until a new command appears, or click the **Add Command** toolbar button. Set the command type for the second command to **Prompt** and type **Hello Again** in the **Message** field.

The Hello World task you just created is a design-time task. It is stored in the project on your local machine. Before you can assign the task to an Agent, you must copy it into the consolidated database by deploying it.

Lesson 11: Deploy a remote task

In the previous lesson you created a remote task. You must now deploy the remote task so that it can be assigned to an Agent.

To deploy a remote task

1. Right-click on the **Hello World** task in the **Folders** view and choose **Deploy**. The **Deploy Remote Task Wizard** is displayed.

2. Accept the defaults on the **Task Name and Destination** page and click **Next**. This gives the deployed task the same name as the design-time task which is what you would normally want to do unless you are deploying the same design-time task for a second time. In that case, you would have to change the name for the deployed task.

3. The **Recipients** page lets you assign the deployed task to existing Agents. You can also do this later as a separate step. From the **Recipients** dropdown, select **Specific Agents**. In the **Agent** list, select **AID_JOHN** and click **Next**.

4. On the **Delivery Options** page, check **The Next Time the Agent Synchronizes** and click **Next**.

5. On the **Reporting Results and Status** page, check **Send Results and Status Immediately** for both questions. This ensures that you receive timely notification when your task executes. For routine tasks

and repetitive tasks you may choose to receive feedback less quickly (especially on success), as this reduces the number of synchronizations of the agent database and the load on the MobiLink server.

6. Click **Finish**. The next time the Agent **AID_JOHN** synchronizes its agent database, it will receive the new task and execute it. Click **OK** on the message boxes with the text **Hello World** and **Hello Again**.

   If you look at the **Folders** view, you can see that there are now two copies of the **Hello World** task in the list. The deployed copy can be seen in the **Folders** view under **Remote Tasks** » **Deployed Tasks**. This is the copy in the consolidated database. The deployed copy of the task can no longer be changed. The design-time copy of the task is still visible under **Remote Tasks**. This task can be changed and can be deployed again with a new name.

   If you want to, you can assign a deployed task to additional Agents at any time by right-clicking on it and choosing **Add Recipients**.

Lesson 12: Check the status of a remote task

Next you can check the status of your remote task in Sybase Central.

To check the status of a remote task

1. In the **Folders** view, click on the deployed version of the **Hello World** task and select the **Results** tab in the right pane. Hit F5 to refresh the tab since task results are not automatically refreshed. On the **Results** tab there is a line for each command in the task, with a **Result Code** that indicates if the command succeeded or failed. A **Result Code** of 0 indicates success.

2. To see the results of a task execution displayed in different ways, select the **Recipients** tab for the deployed task, or look at the **Events** or **Tasks** tab of the Agent that executed the task.

Lesson 13: Create a remote database on a remote device

In this lesson, you use a remote task to create a new remote database on the remote device. If you are adding central administration to an existing synchronization system you can skip this step.

To create a remote database on a remote device

1. The first step is to create a new remote task. In the **Folders** view of Sybase Central under **Central Admin Tutorial**, right-click **Remote Tasks** and choose **New** » **Remote Task**. The **Create Remote Task Wizard** appears.

2. On the **Welcome** page, type **Create DB** in the **Name** field. Unlike the task you created before, this task will create or act on a remote database, so check **This Task Requires or Creates a Remote Database** and select the remote schema name **Tutorial Application v1.0**. This identifies the remote database that the database actions in this task will act on. Click **Next**.

3. On the **Trigger Mechanisms** page, check **When it is Received by an Agent** then click **Finish** to complete the wizard.

4. Click on the newly created **Create DB** task in the **Folders** view.

5. Next, add some commands to the remote task from the **Commands** pane on the right.
   a. The first command creates a new, empty database on the remote device. Set the command type to **Create Database**.
   b. Set the filename to **{db_location}\tutorial_v1.db**. This file name corresponds to the file name in the connection string you specified when you configured the agent.
   c. Press Tab a few times until a new command appears.
   d. The second command creates the schema in the new database. Set the command type to **Execute SQL**. Click **Import**.
   e. From the **Open** window, choose the file *c:\cadmin_demo\remote_db\tutorial_v1.sql* and click **Open**. This imports the SQL for initializing a remote database that was generated when you deployed the synchronization model into the command.

6. The remote task is now complete. Deploy the task and assign it to the agent **AID_JOHN**:
   a. Right-click the **Create DB** task in the **Folders** view and choose **Deploy**. The **Deploy Remote Task Wizard** is displayed.
   b. Accept the defaults on the **Task Name and Destination** page and click **Next**.
   c. From the **Recipients** dropdown, select **Specific Agents**. In the Agent list, select **AID_JOHN** and click **Next**.
   d. On the **Delivery Options** page, check **The Next Time the Agent Synchronizes** and click **Next**.
   e. On the **Reporting Results and Status** page, check **Send Results and Status Immediately** for both questions.
   f. Click **Finish**.

7. Check to see if the task was successful:
   a. In the **Folders** view under **Central Admin Tutorial** » **Consolidated Databases** » **Tutorial** » **Agents**, right-click **AID_JOHN**.
   b. Select the **Events** tab and look for the **Create DB** task. You may need to press F5 to refresh the results.

Lesson 14: Create a synchronization profile

When using central administration you must use a synchronization profile for synchronization. A synchronization profile is a named list of synchronization options that is stored in the remote database. If you are adding central administration to existing synchronization system and your remote database already contains a synchronization profile, you can skip this lesson.

To create a synchronization profile

1. The first step is to create a new remote task. In the **Folders** view of Sybase Central under **Central Admin Tutorial**, right-click **Remote Tasks** and choose **New** » **Remote Task**. The **Create Remote Task Wizard** appears.

2. On the **Welcome** page, type **Create Sync Profile** in the **Name** field. Check **This Task Requires or Creates a Remote Database** and select the remote schema name **Tutorial Application v1.0**. This identifies the remote database that the database actions in this task will act on. Click **Next**.

3. On the **Trigger Mechanisms** page, check **When it is Received by an Agent** then click **Finish** to complete the wizard.

4. Click on the newly created **Create Sync Profile** task in the **Folders** view.

5. Add a single command to the task that executes the necessary SQL to create the synchronization profile. Set the **Command Type** to **Execute SQL**. In the SQL box enter the following:

   ```
   CREATE SYNCHRONIZATION PROFILE normal_sync 'subscription= tutorial1'
   ```

   This creates a synchronization profile named **normal_sync**. The only option specified in the profile is the name of the subscription to synchronize.

6. The remote task is now complete. Deploy the task and assign it to the agent **AID_JOHN**:

   a. Right-click on the **Create Sync Profile** task in the **Folders** view and choose **Deploy**. The **Deploy Remote Task Wizard** is displayed.

   b. Accept the defaults on the **Task Name and Destination** page and click **Next**.

   c. From the **Recipients** dropdown, select **Specific Agents**. In the Agent list, select **AID_JOHN** and click **Next**.

   d. On the **Delivery Options** page, check **The Next Time the Agent Synchronizes** and click **Next**.

   e. On the **Reporting Results and Status** page, check **Send Results and Status Immediately** for both questions.

   f. Click **Finish**.

Lesson 15: Schedule synchronization

The next step is to configure the MobiLink Agent to synchronize its remote database at regular intervals. You do this by creating a remote task that executes based on a schedule, and synchronizes the database each time it executes. This task is different from the other tasks you've created because the other tasks executed only once. This tasks remains on the remote device and executes at regular intervals until you stop it.

To schedule synchronization

1. The first step is to create a new remote task. In the **Folders** view of Sybase Central under **Central Admin Tutorial**, right-click **Remote Tasks** and choose **New** » **Remote Task**. The **Create Remote Task Wizard** appears.
2. On the **Welcome** page, type **Sync** in the **Name** field. Check **This Task Requires or Creates a Remote Database** and select the remote schema name **Tutorial Application v1.0**. This identifies the remote database that the database actions in this task will act on. Click **Next**.
3. On the **Trigger Mechanisms** page, check **Based on a Schedule** then click **Next**.
4. Accept the defaults on the **Start Time and Date** page. This allows the task to start running immediately. Click **Next**.
5. On the **Repetition** page, check **Repeat Every** and set the interval to one minute. Click **Finish** to complete the wizard.
6. Click on the newly created **Sync** task in the **Folders** view.
7. Add a single command to the task that will cause a synchronization.
   a. On the **Commands** tab, set the **Command Type** for the first command to **Synchronize**.
   b. For **Synchronization Profile**, type **normal_sync**. This is the synchronization profile you created when you created the remote database.
8. The synchronization task is now complete. Right-click **Sync** and choose **Deploy**. Click **Next**.
9. From the **Recipients** dropdown, choose **Specific Agents** and assign the task to agent **AID_JOHN**. Click **Next** then **Next** again.
10. On the **Reporting Results and Status** page, set **If Task Succeeds** to **Send Only Status Later** and set **If Task Fails** to **Send Results and Status Immediately**.

    Since this task repeats frequently, it is a good idea to limit the feedback requested in order to improve performance.
11. Click **Finish**. Once the Agent receives this new task, it begins to synchronize its remote database once each minute.

Lesson 16: Modify scheduled synchronizations

In the last lesson you created a remote task to synchronize the remote database once per minute. In this lesson, you change the synchronization interval to once per hour.

Once a remote task is deployed, the deployed version cannot be modified. Instead, you create a new remote task with the desired modifications, then you cancel the existing task and deploy the new task to replace it.

To modify a scheduled synchronization

1. First, create a new remote task with the desired repeat interval using the existing deployed task as a template.
   a. In the **Folders** view, under **Central Admin Tutorial** » **Remote Tasks** » **Deployed Tasks** right click **Sync** and select **Copy** to copy the task to the clipboard.
   b. Right-click **Remote Tasks** and select **Paste**. A dialog appears asking for you to rename the remote task. Type **Sync every hour** and click **OK**.
   c. Right-click on the new **Sync every hour** task and select **Properties**. On the **Repetition** page of the property sheet, change the **Repeat Every** value from **1 minutes** to **1 hours** and click **OK**.
2. Next, cancel the existing remote task that causes synchronization each minute.
   a. In the **Folders** view, click on the deployed version of the **Sync** task and select the **Recipients** tab.
   b. Right-click on the entry in the table for agent **AID_JOHN** and select **Cancel**.
3. Lastly, deploy the new **Sync every hour** task and assign it to agent **AID_JOHN**.
   a. Right-click **Sync every hour** and choose **Deploy**. Click **Next**.
   b. From the **Recipients** dropdown, choose **Specific Agents** and assign the task to agent **AID_JOHN**. Click **Next** then **Next** again.
   c. On the **Reporting Results and Status** page, set **If Task Succeeds** to **Send Only Status Later** and set **If Task Fails** to **Send Results and Status Immediately**.
   d. Click **Finish**. Once the Agent receives this new task, it begins to synchronize its remote database once an hour instead of once a minute.

Lesson 17: Force immediate synchronization

In the last lesson, you set up the remote database to synchronize once per hour. This lesson shows you how to use a server-initiated remote task (SIRT) to force a synchronization before the hour is up. This technique is useful whenever you want to centrally control when a certain task executes.

To force immediate synchronization using SIRT

1. From the **Folders** view under **Central Admin Tutorial** » **Remote Tasks** » **Deployed Tasks** » , right-click **Sync every hour** and choose **Initiate**.
2. On the **Initiate Remote Task** dialog select **All Agents** and click **OK**.

   All recipients of the task are instructed to execute the task immediately, the next time they poll the server. The frequency with which Agents poll the server is controlled by the **Administration Polling Interval** property of the Agent.

Lesson 18: Change the remote schema

In this lesson, you change the schema of the remote database. For our purposes, a schema change occurs whenever we change the remote schema name of the database. You are never forced to change the remote schema name, it is always left up to your discretion.

You should try to ensure that any remote task that you can execute against one remote database can be executed against any other remote database with the same remote schema name. You should change a database's remote schema name whenever you change the database in a way that would make a task fail or succeed. The only commands within a task that are affected by the state of the remote database are the **Synchronize** and **Execute SQL** commands.

Synchronize commands depend on the presence of synchronization profiles in the remote database, so you should always change remote schema names when you add or remove a synchronization profile.

Execute SQL commands depend on the state of many database objects that you would normally consider to be schema. Some examples of changes that would affect **Execute SQL** commands, and hence require a remote schema name change, are adding or remove tables from the database, altering the definition of tables in the database, and adding or removing stored procedures.

In this tutorial, you alter the schema of the remote database by adding a new table to it.

To change a remote schema

1. Return to the **Folders** view in Sybase Central. Under **Central Admin Tutorial**, right-click **Remote Schema Names** and choose **New** » **Remote Schema Name**. The **Create Remote Schema Name Wizard** appears.
2. Type **Tutorial Application v2.0** for the schema name.
3. Select **SQL Anywhere** as the database type and click **Finish**.
4. The next step is to create a new remote task. In the **Folders** view of Sybase Central under **Central Admin Tutorial**, right-click **Remote Tasks** and choose **New** » **Remote Task**. The **Create Remote Task Wizard** appears.
5. On the **Welcome** page, type **Schema Upgrade** in the **Name** field.
6. Check **This Task Requires or Creates a Remote Database** and set the **Remote Schema Name** to **Tutorial Application v1.0**.
7. Check **This Task Upgrades the Schema of the Managed Remote Database** and set **New Remote Schema Name** to **Tutorial Application v2.0**. Click **Finish**.
8. On the **Commands** tab, select **Execute SQL** from the **Command Type** dropdown list. In the **SQL** field, type the following:

```
CREATE TABLE  product (
    prod_id            integer primary key,
    name               varchar( 100 )
)
```


The schema change task is now complete.

Before you deploy the new schema change task, you must consider any tasks already assigned to the remote device. After the **Schema Upgrade** task completes, the remote schema name for the database will be **Tutorial Application v2.0**. Any tasks on the remote device that are associated with the old remote schema name, **Tutorial Application v1.0** will no longer be able to run and will be discarded by the Agent. In order to maintain the functionality provided by these tasks, you must create new versions of the tasks and associate them with the new remote schema name.

9. In the **Folders** view, under **Central Admin Tutorial** » **Consolidated Databases** » **Tutorial** » **Agents** » click **AID_JOHN**. Select the **Tasks** tab in the right pane. Only active tasks are still being executed by the Agent. These are the only tasks that you may need to create new versions of. In this case, the only active task is the **Sync every hour** task.

   You can determine if this task is associated with the old remote schema name by checking the **Remote Schema Name** column on the **Tasks** tab. This tasks shows that the **Remote Schema Name** of the **Sync every hour** task is **Tutorial Application v1.0** so it is associated with the old remote schema name. In order to continue synchronization after the schema change, you need to create a new version of this task and assign it to the Agent.

10. Right-click the **Sync every hour** task and choose **Go to Task**.

11. Right-click the deployed task **Sync every hour** and select **Copy**.

12. Right-click **Remote Tasks** and choose **Paste**. When asked for a name for the copied task, type **Sync every hour v2** and click **OK**.

13. Next, you need to consider whether commands in the task require any changes to continue working with the new schema. In this case, the answer is no. There is only 1 command and it only depends on the **normal_sync** synchronization profile, which you have not modified with this schema change.

14. You must now mark the task as being associated with the new remote schema name. Right-click the **Sync every hour v2** task and select **Properties**. On the **General** page of the property sheet, select **Tutorial Application v2.0** for the **Remote Schema Name** and click **OK**.

15. To deploy the new task, right-click the **Sync every hour v2** task and choose **Deploy**. Click **Next**.

16. For **Recipients**, choose **Specific Agents** and then select agent **AID_JOHN**. Click **Next** then **Finish**.

17. Right-click the **Schema Upgrade** task and choose **Deploy**. Click **Next**.

18. From the **Recipients** dropdown, choose **Specific Agents** and assign the task to agent **AID_JOHN**. Click **Next** then **Finish**.

    You should see the **Schema Upgrade** task execute successfully. After that, the **Sync every hour v2** task should start executing each hour and the **Sync every hour** task should stop executing.

Lesson 19: Query the remote database

In this lesson, you query the remote database and return results to the server. This is very useful when troubleshooting because you can find out exactly what state the remote database is in.

The tables you've added to the database in this tutorial do not contain any data, so instead you'll query the database system tables. Even though you are querying a system table in this example, everything you do works exactly the same way if you queried a user table.

Suppose that you wanted to confirm that the schema change you performed in the last lesson did what you expected, that the product table was created with the correct columns. You could confirm that by querying the systable and systabcol system tables.

To query the remote database

1. In the **Folders** view of Sybase Central under **Central Admin Tutorial**, right-click **Remote Tasks** and choose **New** » **Remote Task**. The **Create Remote Task Wizard** appears.
2. On the **Welcome** page, type **Table Query** in the **Name** field.
3. Check **This Task Requires or Creates a Remote Database** and set the **Remote Schema Name** to **Tutorial Application v2.0** and click **Next**.
4. On the **Trigger Mechanisms** page, check **When it is Received by an Agent** and click **Finish**.
5. Add an **Execute SQL** command to the task with the following SQL:

   ```
   select * from systable where table_name = 'product'
   go
   select * from systabcol order by table_id
   ```

6. Right-click the new **Table Query** task and choose **Deploy**. Click **Next**.
7. For **Recipients**, choose **Specific Agents**, select agent **AID_JOHN** and click **Next** then **Next** again.
8. On the **Reporting Results and Status** page, set both **If Task Succeeds** and **If Task Fails** to **Send Results and Status Immediately**. Click **Finish** and wait until the task executes.
9. Click the deployed copy of the **Table Query** task in the **Folders** view and select the **Results** tab. If you don't see any results on the tab press, F5 to refresh it.
10. Right-click on the line in the table for the **Execute SQL** statement and select **Details**. The **Command Result** dialog appears.
11. Pick the **Results** tab on the dialog. This tab shows results of any queries executed. Press F5 to refresh the results as necessary. The **Result** dropdown at the top of the pane allows you to switch between results for the two queries.

Lesson 20: Upload files using SIRT

In this lesson, you upload files from the remote device using a server-initiated remote task (SIRT). This is a useful troubleshooting technique because an administrator can examine the files for problems.

When you started the MobiLink Agent on the remote device, you directed it to log messages to the file *agent.txt*. You are now going to retrieve and examine that file from the remote device.

To upload files

1. In the **Folders** view of Sybase Central under **Central Admin Tutorial**, right-click **Remote Tasks** and choose **New** » **Remote Task**. The **Create Remote Task Wizard** appears.

2. On the **Welcome** page, type **Upload Agent Log** in the **Name** field.

3. Uncheck **This Task Requires or Creates a Remote Database** if it is selected and click **Next**.

4. On the **Trigger Mechanisms** page, click **Next**. This task is going to be defined as a server-initiated remote task and will be triggered by the administrator in Sybase Central. Click **Finish**.

5. Click the new task in the **Folders** view and add a command to the task. Set the **Command Type** to **Upload File**.

6. Set the **Server File Name** to *{agent_id}\agent.txt* and the **Remote File Name** to *{agent_log}*. You can use the ellipsis (three dots) button in the command editor to easily enter the macro values.

   The *{agent_log}* macro is replaced by the name of the log file being kept by the MobiLink Agent on the remote device.

   Notice that in the **Server File Name** field you specified the directory where the file will be placed using the *{agent_id}* macro. This is very important. If you do not use a macro when specifying the server file name then every Agent that executes the task will place their upload file in the same place, and each new Agent will overwrite the file written by the previous agent. By using a macro, we ensure that each agent will upload its log file to a different location on the server, allowing us to view all the log files.

7. Right-click the new **Upload Agent Log** task and choose **Deploy**. Click **Next**.

8. For **Recipients**, choose **Specific Agents** and then select agent **AID_JOHN**. Click **Next**.

9. On the **Delivery Options** page, choose **The Next Time the Agent Synchronizes** and click **Next**.

10. On the **Reporting Results and Status** page, set both **If Task Succeeds** and **If Task Fails** to **Send Results and Status Immediately**. Click **Finish**.

11. The task needs to be Initiated by the administrator in Sybase Central. To Initiate the task, go to *AID_JOHN* under the **Agents**. In the pane on the right, select the **Task** tab. In the **Task** tab, right-click on the **Upload Agent Log** task and choose **Initiate**. Wait for the task to execute.

    The uploaded file is placed in the MobiLink upload directory that was specified it with the -ftru option on the MobiLink command line. You specified *c:\cadmin_demo\consolidated\upload* for the upload directory. Take a look at that directory using a command prompt or the Windows Explorer. You should find the *AID_JOHN* subdirectory. In that subdirectory is the *agent.txt* file that you uploaded. To upload the Agent log again, repeat step 11.

Tutorial: Changing a schema using the script version clause

Introduction to schema change tutorial

This tutorial describes how to perform a schema change on a remote database involved in synchronization where the dbmlsync ScriptVersion extended option is not being used. In this tutorial, you set up a synchronization system that synchronizes a single table, and then make a schema change to add a column to the synchronizing table and continue synchronizing.

Required software

This tutorial assumes you have a complete install of SQL Anywhere, including MobiLink on your local computer where you are running the tutorial.

Overview

This tutorial shows you how to:

Suggested background reading

- "Introduction to MobiLink client schema changes" [*MobiLink - Client Administration*]

Lesson 1: Create and configure the consolidated database

In this lesson, you set up a consolidated database for synchronization.

To create and configure a consolidated database

1. Run the following commands to create a consolidated database and start it running:

```
md c:\cons
cd c:\cons
```

```
dbinit consol.db
dbeng12 consol.db
```

2. Run the following command to define an ODBC data source for the consolidated database:

```
dbdsn -w dsn_consol -y -c "uid=DBA;pwd=sql;dbf=consol.db;server=consol"
```

3. To use a database as a consolidated database, you must run a setup script that adds system tables, views, and stored procedures that are used by MobiLink. The following command sets up *consol.db* as a consolidated database:

```
dbisql -c "dsn=dsn_consol" %SQLANY12%\MobiLink\setup\syncsa.sql
```

4. Open Interactive SQL and connect to *consol.db* using the **dsn_consol** DSN.

```
dbisql -c "dsn=dsn_consol"
```

5. Run the following SQL statements. They create the customer table on the consolidated database and create the required synchronization scripts.

```
CREATE TABLE customer (
   id      unsigned integer primary key,
   name    varchar( 256),
   phone   varchar( 12 )
);

CALL ml_add_column('my_ver1', 'customer', 'id', null );
CALL ml_add_column('my_ver1', 'customer', 'name', null );
CALL ml_add_column('my_ver1', 'customer', 'phone', null );

CALL ml_add_table_script( 'my_ver1', 'customer', 'upload_insert',
      'INSERT INTO customer ( id, name, phone ) '
      || 'VALUES ({ml r.id}, {ml r.name}, {ml r.phone} )' );

CALL ml_add_table_script( 'my_ver1', 'customer', 'download_cursor',
       'SELECT id, name, phone from customer' );

CALL ml_add_table_script( 'my_ver1', 'customer', 'download_delete_cursor',
'--{ml_ignore}' );
COMMIT;
```

   After you have executed the SQL, leave Interactive SQL running and connected to the database as you will be running more SQL against the database as you work through the tutorial.

6. Start the MobiLink server by running the following command:

```
start mlsrv12 -c "dsn=dsn_consol" -v+ -ot mlsrv.txt -zu+
```

Lesson 2: Create and configure the remote database

In this lesson, you set up a remote database for synchronization.

To create and configure a remote database

1. Run the following commands to create a remote database and start it running:

```
cd..
md c:\remote
```



```
cd c:\remote
dbinit remote.db
dbeng12 remote.db
```

2. Open another instance of Interactive SQL and connect to *remote.db*.

```
dbisql -c "eng=remote;dbf=remote.db;uid=dba;pwd=sql"
```

3. Run the following SQL in Interactive SQL to create objects in the remote database.

   First, create the table to be synchronized.

```
CREATE TABLE customer (
   id      unsigned integer primary key,
   name    varchar( 256),
   phone   varchar( 12 )
);
```

4. Still using the Interactive SQL instance connected to the remote database, create a publication, MobiLink user and subscription. Notice that the script version is associated with the subscription using the SCRIPT VERSION clause. This is very important since the schema upgrade procedure shown in this tutorial only works for subscriptions that have the script version set using the SCRIPT VERSION clause.

```
CREATE PUBLICATION p1 (
      TABLE customer
);

CREATE SYNCHRONIZATION USER u1;

CREATE SYNCHRONIZATION SUBSCRIPTION my_sub
TO p1
FOR u1
SCRIPT VERSION 'my_ver1';
```

After you have executed the SQL, leave Interactive SQL running and connected to the database as you will be running more SQL against the database as you work through the tutorial.

Lesson 3: Synchronize the remote database

You should now have a working synchronization system set up. In this lesson, you test it by inserting some data and synchronizing.

To synchronize the remote database

1. Using the instance of Interactive SQL that's connected to the consolidated database, execute the following SQL to insert a row in the customer table.

```
INSERT INTO customer VALUES( 100, 'John Jones', '519-555-1234' );
COMMIT;
```

2. Using the instance of Interactive SQL that's connected to the remote database, execute the following SQL to insert a row in the customer table.

```
INSERT INTO customer VALUES( 1, 'Willie Lowman', '705-411-6372' );
COMMIT;
```

3. Now synchronize by running the following command:

```
dbmlsync -v+ -ot sync1.txt -c uid=dba;pwd=sql;eng=remote -s my_sub -k
```

You can confirm that the synchronization succeeded by comparing the contents of the customer table in the remote and consolidated databases. You might also want to look at the dbmlync log, *sync1.txt* and check for errors.

Lesson 4: Insert data in the remote database

In this lesson, you insert data into the remote database to demonstrate that a schema change can proceed even if there are operations in the remote database that need to be uploaded.

To insert data in the remote database

- Using the instance of Interactive SQL that's connected to the remote database, execute the following SQL to insert a row in the customer table:

```
INSERT INTO customer VALUES( 2, 'Sue Slow', '602-411-5467' );
COMMIT;
```

Lesson 5: Perform a schema change on the consolidated database

In this lesson, you perform a schema change on the consolidated database.

To perform a schema change on the consolidated database

1. Now you are going to add a new column to the customer table to store the customer's cell phone number. First, add the new column to the consolidated database by executing the following SQL in the instance of Interactive SQL that's connected to the consolidated database.

```
ALTER TABLE customer ADD cell_phone varchar(12) default null;
```

2. Next, create a new script version called **my_ver2** to handle synchronizations from remote databases with the new schema. Remote databases with the old schema continue to use the old script version, **my_ver1**. Execute the following SQL against the consolidated database:

```
CALL ml_add_column('my_ver2', 'customer', 'id', null );
CALL ml_add_column('my_ver2', 'customer', 'name', null );
CALL ml_add_column('my_ver2', 'customer', 'phone', null );
CALL ml_add_column('my_ver2', 'customer', 'cell_phone', null );

CALL ml_add_table_script( 'my_ver2', 'customer', 'upload_insert',
      'INSERT INTO customer ( id, name, phone, cell_phone ) '
      || 'VALUES ({ml r.id}, {ml r.name}, {ml r.phone}, {ml
r.cell_phone})' );

CALL ml_add_table_script( 'my_ver2', 'customer', 'download_cursor',
```



```
        'SELECT id, name, phone, cell_phone from customer' );
CALL ml_add_table_script( 'my_ver2', 'customer', 'download_delete_cursor',
'--{ml_ignore}' );
COMMIT;
```

Lesson 6: Perform a schema change on the remote database

In this lesson, you modify the remote database to add the new column to the customer table and to change the script version used to synchronize.

To perform a schema change on the remote database

1. First, you start a synchronization schema change. This is required for most schema changes that affect synchronizing tables. This statement changes the script version that is used to synchronize the subscription, and locks the affected table so the schema change can proceed safely.

   Execute the following SQL on the remote database using the instance of Interactive SQL that's connected to the remote database:

   ```
   START SYNCHRONIZATION SCHEMA CHANGE
   FOR TABLES customer
   SET SCRIPT VERSION = 'my_ver2';
   ```

2. Next, add the new column to the customer table with the following SQL:

   ```
   ALTER TABLE customer ADD cell_phone varchar(12) default null;
   ```

3. Lasting, close the schema change which unlocks the tables.

   ```
   STOP SYNCHRONIZATION SCHEMA CHANGE;
   ```

Lesson 7: Insert data in the remote database

In this lesson, you insert some more data into the remote and consolidated databases using the new schema.

To insert data in the remote database

1. Using Interactive SQL, execute the following SQL against the remote database:

   ```
   INSERT INTO customer VALUES( 3, 'Mo Hamid', '613-411-9999',
   '613-502-1212' );
   COMMIT;
   ```

2. Using Interactive SQL, execute the following SQL against the consolidated database:

   ```
   INSERT INTO customer VALUES( 101, 'Theo Tug', '212-911-7677',
   '212-311-3900' );
   COMMIT;
   ```

Lesson 8: Synchronize

In this lesson, you synchronize again with the schema changes.

To synchronize

- Synchronize again by running the following command:

  ```
  dbmlsync -v+ -ot sync2.txt -c uid=dba;pwd=sql;eng=remote -s my_sub -k
  ```

  The row for **Sue Slow** that was inserted before the schema change is uploaded using the script version **my_ver1**. The row for **Mo Hamid** that was inserted after the schema change is uploaded using the script version **my_ver2**. Rows are downloaded using the download cursor for **my_ver2**.

The schema change is now complete and you can continue synchronizing normally.

Tutorial: Changing a schema using the ScriptVersion extended option

Introduction to changing a schema using the ScriptVersion extended option

This tutorial demonstrates how to perform a schema change when you are using the ScriptVersion extended option.

> **Note**
> It is recommended that you avoid using the ScriptVersion extended option if possible. Instead associate your script version with your subscription using the SCRIPT VERSION clause on the CREATE SYNCHRONIZATION SUBSCRIPTION statement or the SET SCRIPT VERSION clause on the ALTER SYNCHRONIZATION SUBSCRIPTION statement. This will give you more flexibility to perform schema upgrades.

Required software

This tutorial assumes you have a complete install of SQL Anywhere, including MobiLink on your local computer where you are running the tutorial.

Overview

This tutorial shows you how to:

Lesson 1: Create and configure the consolidated database

In this lesson, you set up a consolidated database for synchronization.

To create and configure a consolidated database

1. Run the following commands to create a consolidated database and start it running:

```
md c:\cons
cd c:\cons
dbinit consol.db
dbeng12 consol.db
```

2. Run the following command to define an ODBC data source for the consolidated database:

```
dbdsn -w dsn_consol -y -c "uid=DBA;pwd=sql;dbf=consol.db;server=consol"
```

3. To use a database as a consolidated database, you must run a setup script that adds system tables, views, and stored procedures that are used by MobiLink. The following command sets up *consol.db* as a consolidated database:

```
dbisql -c "dsn=dsn_consol" %SQLANY12%\MobiLink\setup\syncsa.sql
```

4. Open Interactive SQL and connect to *consol.db* using the **dsn_consol** DSN.

```
dbisql -c "dsn=dsn_consol"
```

5. Run the following SQL statements in Interactive SQL. They create the customer table on the consolidated database and create the required synchronization scripts.

```
CREATE TABLE customer (
   id      unsigned integer primary key,
   name    varchar( 256),
   phone   varchar( 12 )
);

CALL ml_add_column('my_ver1', 'customer', 'id', null );
CALL ml_add_column('my_ver1', 'customer', 'name', null );
CALL ml_add_column('my_ver1', 'customer', 'phone', null );

CALL ml_add_table_script( 'my_ver1', 'customer', 'upload_insert',
      'INSERT INTO customer ( id, name, phone ) '
      || 'VALUES ({ml r.id}, {ml r.name}, {ml r.phone} )' );

CALL ml_add_table_script( 'my_ver1', 'customer', 'download_cursor',
       'SELECT id, name, phone from customer' );

CALL ml_add_table_script( 'my_ver1', 'customer', 'download_delete_cursor',
'--{ml_ignore}' );
COMMIT;
```

   After you have executed the SQL, leave Interactive SQL running and connected to the database as you will be running more SQL against the database as you work through the tutorial.

6. Start the MobiLink server by running the following command:

```
start mlsrv12 -c "dsn=dsn_consol" -v+ -ot mlsrv.txt -zu+
```

Lesson 2: Create and configure the remote database

In this lesson, you set up a remote database for synchronization.

To create and configure a remote database

1. Run the following commands to create a remote database and start it running:

```
cd..
md c:\remote
cd c:\remote
dbinit remote.db
dbeng12 remote.db
```

2. Open another instance of Interactive SQL and connect to *remote.db*.

```
dbisql -c "eng=remote;dbf=remote.db;uid=dba;pwd=sql"
```

3. Run the following SQL in Interactive SQL to create objects in the remote database.

   First, create the table to be synchronized.

```
CREATE TABLE customer (
   id      unsigned integer primary key,
   name    varchar( 256),
   phone   varchar( 12 )
);
```

4. Next, create a publication, MobiLink user and subscription.

```
CREATE PUBLICATION p1 (
   TABLE customer
);

CREATE SYNCHRONIZATION USER u1;

CREATE SYNCHRONIZATION SUBSCRIPTION my_sub
TO p1
FOR u1
OPTION ScriptVersion='my_ver1';
```

After you have executed the SQL, leave Interactive SQL running and connected to the database as you will be running more SQL against the database as you work through the tutorial.

Lesson 3: Synchronize the remote database

You should now have a working synchronization system set up. In this lesson, you test it by inserting some data and synchronizing.

To synchronize the remote database

1. Using the instance of Interactive SQL that's connected to the consolidated database, execute the following SQL to insert a row in the customer table.


```
INSERT INTO customer VALUES( 100, 'John Jones', '519-555-1234' );
COMMIT;
```

2. Using the instance of Interactive SQL that's connected to the remote database, execute the following SQL to insert a row in the customer table.

```
INSERT INTO customer VALUES( 1, 'Willie Lowman', '705-411-6372' );
COMMIT;
```

3. Now synchronize by running the following command:

```
dbmlsync -v+ -ot sync1.txt -c uid=dba;pwd=sql;eng=remote -s my_sub -k
```

You can confirm that the synchronization succeeded by comparing the contents of the customer table in the remote and consolidated databases. You might also want to look at the dbmlync log, *sync1.txt* and check for errors.

Lesson 4: Perform schema change on the consolidated database

In this lesson, add a new column to the customer table to store the customer's cell phone number.

To perform a schema change on the consolidated database

1. Using the instance of Interactive SQL that's connected to the consolidated database, execute the following SQL to insert a row in the customer table:

```
ALTER TABLE customer ADD cell_phone varchar(12) default null;
```

2. Next, create a new script version called my_ver2 to handle synchronizations from remote databases with the new schema. Remote databases with the old schema will continue to use the old script version, my_ver1.

   Execute the following SQL against the consolidated database:

```
CALL ml_add_column('my_ver2', 'customer', 'id', null );
CALL ml_add_column('my_ver2', 'customer', 'name', null );
CALL ml_add_column('my_ver2', 'customer', 'phone', null );
CALL ml_add_column('my_ver2', 'customer', 'cell_phone', null );

CALL ml_add_table_script( 'my_ver2', 'customer', 'upload_insert',
      'INSERT INTO customer ( id, name, phone, cell_phone ) '
      || 'VALUES ({ml r.id}, {ml r.name}, {ml r.phone}, {ml
r.cell_phone})' );

CALL ml_add_table_script( 'my_ver2', 'customer', 'download_cursor',
       'SELECT id, name, phone, cell_phone from customer' );

CALL ml_add_table_script( 'my_ver2', 'customer', 'download_delete_cursor',
'--{ml_ignore}' );

COMMIT;
```

Lesson 5: Perform a schema change on the remote database

In this lesson, you modify the remote database to add the new column to the customer table and to change the script version used to synchronize. Before you do this you must ensure that there are no operations for the customer table that need to be uploaded. The best way to do this is to perform the schema change in the sp_hook_dbmlsync_schema_upgrade hook. When you use this hook, dbmlsync ensures that the schema change is performed safely by locking the synchronizing tables at the start of synchronization and holding the locks until the schema change is complete.

Caution
If you change the schema when there are operations to be uploaded, the remote is always unable to synchronize after the schema change.

To perform a schema change on the remote database

1. Create an sp_hook_dbmlsync_schema_upgrade hook by executing the following SQL against the remote database. The hook adds a new column to the customer table and changes the value of the ScriptVersion extended option stored with the subscription. The hook is deleted by dbmlsync after it has executed.

```
CREATE PROCEDURE sp_hook_dbmlsync_schema_upgrade()
BEGIN
    ALTER TABLE customer
    ADD cell_phone varchar(12) default null;

    ALTER SYNCHRONIZATION SUBSCRIPTION my_sub
    ALTER OPTION ScriptVersion='my_ver2';

    UPDATE #hook_dict
    SET value = 'always'
        WHERE name = 'drop hook';
END
```

2. Next, synchronize to upload any operations that need to be uploaded and to perform the schema change by executing the sp_hook_dbmlsync_schema_change hook. Run the following command:

```
dbmlsync -v+ -ot sync2.txt -c uid=dba;pwd=sql;eng=remote -s my_sub -k
```

After this synchronization, it is a very good idea to look at the dbmlsync log *sync2.txt* to ensure that there are no errors to indicate that the schema change was not completed.

The schema change is now complete and you can continue synchronizing normally.

Index

Symbols

A

B

C

D

E

N

O

P

Q

R

S

T

U

V

W

