MobiLink Getting Started

Published: October 2006

Copyright and trademarks

Contents

About This Manual

Subject

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

Audience

This manual is for users of SQL Anywhere and other relational database systems who want to add synchronization to their information systems.

Before you begin

☞ For a comparison of MobiLink with other synchronization and replication technologies, see "Overview of Data Exchange Technologies" [*SQL Anywhere 10 - Introduction*].

SQL Anywhere documentation

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

The SQL Anywhere documentation

The complete SQL Anywhere documentation is available in two forms: an online form that combines all books, and as separate PDF files for each book. Both forms of the documentation contain identical information and consist of the following books:

- ♦ **SQL Anywhere 10 - Introduction** This book introduces SQL Anywhere 10—a comprehensive package that provides data management and data exchange, enabling the rapid development of database-powered applications for server, desktop, mobile, and remote office environments.
- ♦ **SQL Anywhere 10 - Changes and Upgrading** This book describes new features in SQL Anywhere 10 and in previous versions of the software.
- ♦ **SQL Anywhere Server - Database Administration** This book covers material related to running, managing, and configuring SQL Anywhere databases. It describes database connections, the database server, database files, security, backup procedures, security, and replication with Replication Server, as well as administration utilities and options.
- ♦ **SQL Anywhere Server - SQL Usage** This book describes how to design and create databases; how to import, export, and modify data; how to retrieve data; and how to build stored procedures and triggers.
- ♦ **SQL Anywhere Server - SQL Reference** This book provides a complete reference for the SQL language used by SQL Anywhere. It also describes the SQL Anywhere system views and procedures.
- ♦ **SQL Anywhere Server - Programming** This book describes how to build and deploy database applications using the C, C++, and Java programming languages, as well as Visual Studio .NET. Users of tools such as Visual Basic and PowerBuilder can use the programming interfaces provided by those tools.
- ♦ **SQL Anywhere 10 - Error Messages** This book provides a complete listing of SQL Anywhere error messages together with diagnostic information.
- ♦ **MobiLink - Getting Started** This manual introduces MobiLink, a session-based relational-database synchronization system. MobiLink technology allows two-way replication and is well suited to mobile computing environments.
- ♦ **MobiLink - Server Administration** This manual describes how to set up and administer MobiLink applications.
- ♦ **MobiLink - Client Administration** This manual describes how to set up, configure, and synchronize MobiLink clients. MobiLink clients can be SQL Anywhere or UltraLite databases.
- ♦ **MobiLink - Server-Initiated Synchronization** This manual describes MobiLink server-initiated synchronization, a feature of MobiLink that allows you to initiate synchronization or other remote actions from the consolidated database.

- **QAnywhere** This manual describes QAnywhere, which defines a messaging platform for mobile and wireless clients as well as traditional desktop and laptop clients.
- **SQL Remote** This book describes the SQL Remote data replication system for mobile computing, which enables sharing of data between a SQL Anywhere consolidated database and many SQL Anywhere remote databases using an indirect link such as email or file transfer.
- **SQL Anywhere 10 - Context-Sensitive Help** This manual provides context-sensitive help for the Connect dialog, the Query Editor, the MobiLink Monitor, the SQL Anywhere Console utility, the Index Consultant, and Interactive SQL.
- **UltraLite - Database Management and Reference** This manual introduces the UltraLite database system for small devices.
- **UltraLite - AppForge Programming** This manual describes UltraLite for AppForge. With UltraLite for AppForge you can develop and deploy database applications to handheld, mobile, or embedded devices, running Palm OS, Symbian OS, or Windows CE.
- **UltraLite - .NET Programming** This manual describes UltraLite.NET. With UltraLite.NET you can develop and deploy database applications to computers, or handheld, mobile, or embedded devices.
- **UltraLite - M-Business Anywhere Programming** This manual describes UltraLite for M-Business Anywhere. With UltraLite for M-Business Anywhere you can develop and deploy web-based database applications to handheld, mobile, or embedded devices, running Palm OS, Windows CE, or Windows XP.
- **UltraLite - C and C++ Programming** This manual describes UltraLite C and C++ programming interfaces. With UltraLite you can develop and deploy database applications to handheld, mobile, or embedded devices.

Documentation formats

SQL Anywhere provides documentation in the following formats:

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

  To access the online documentation on Windows operating systems, choose Start ► Programs ► SQL Anywhere 10 ► Online Books. You can navigate the online documentation using the HTML Help table of contents, index, and search facility in the left pane, as well as using the links and menus in the right pane.

  To access the online documentation on Unix operating systems, see the HTML documentation under your SQL Anywhere installation or on your installation CD.
- **PDF files** The complete set of SQL Anywhere books is provided as a set of Adobe Portable Document Format (pdf) files, viewable with Adobe Reader.

  On Windows, the PDF books are accessible from the online books via the PDF link at the top of each page, or from the Windows Start menu (Start ► Programs ► SQL Anywhere 10 ► Online Books - PDF Format).

On Unix, the PDF books are accessible on your installation CD.

Documentation conventions

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

Syntax conventions

The following conventions are used in the SQL syntax descriptions:

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

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

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

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

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

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

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

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

  RELEASE SAVEPOINT [*savepoint-name*]

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

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

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

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

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

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

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

File name conventions

The documentation generally adopts Windows conventions when describing operating-system dependent tasks and features such as paths and file names. In most cases, there is a simple transformation to the syntax used on other operating systems.

- **Directories and path names** The documentation typically lists directory paths using Windows conventions, including colons for drives and backslashes as a directory separator. For example,

  ```
  MobiLink\redirector
  ```

  On Unix, Linux, and Mac OS X, you should use forward slashes instead. For example,

  ```
  MobiLink/redirector
  ```

- **Executable files** The documentation shows executable file names using Windows conventions, with the suffix *.exe*. On Unix, Linux, and Mac OS X, executable file names have no suffix. On NetWare, executable file names use the suffix *.nlm*.

  For example, on Windows, the network database server is *dbsrv10.exe*. On Unix, Linux, and Mac OS X, it is *dbsrv10*. On NetWare, it is *dbsrv10.nlm*.

- **install-dir** The installation process allows you to choose where to install SQL Anywhere, and the documentation refers to this location using the convention *install-dir*.

  After installation is complete, the environment variable SQLANY10 specifies the location of the installation directory containing the SQL Anywhere components (*install-dir*). SQLANYSH10 specifies the location of the directory containing components shared by SQL Anywhere with other Sybase applications.

  For more information on the default location of *install-dir*, by operating system, see "File Locations and Installation Settings" [*SQL Anywhere Server - Database Administration*].

- **samples-dir** The installation process allows you to choose where to install the samples that are included with SQL Anywhere, and the documentation refers to this location using the convention *samples-dir*.

  After installation is complete, the environment variable SQLANYSAMP10 specifies the location of the directory containing the samples (*samples-dir*). From the Windows Start menu, choosing Programs ▶ SQL Anywhere 10 ▶ Sample Applications and Projects opens a Windows Explorer window in this directory.

  For more information on the default location of *samples-dir*, by operating system, see "The samples directory" [*SQL Anywhere Server - Database Administration*].

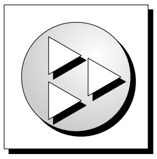
- **Environment variables** The documentation refers to setting environment variables. On Windows, environment variables are referred to using the syntax *%envvar%*. On Unix, Linux, and Mac OS X, environment variables are referred to using the syntax *$envvar* or *${envvar}*.

Unix, Linux, and Mac OS X environment variables are stored in shell and login startup files, such as *.cshrc* or *.tcshrc*.

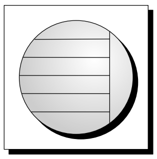
Graphic icons

The following icons are used in this documentation.

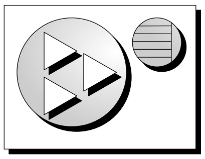
♦ A client application.



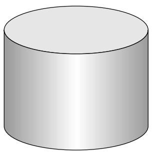
♦ A database server, such as SQL Anywhere.



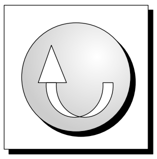
♦ An UltraLite application.



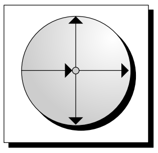
♦ A database. In some high-level diagrams, the icon may be used to represent both the database and the database server that manages it.



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



♦ A Sybase Replication Server



- A programming interface.

Finding out more and providing feedback

Finding out more

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

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

When you write to one of these newsgroups, always provide detailed information about your problem, including the build number of your version of SQL Anywhere. You can find this information by entering **dbeng10 -v** at a command prompt.

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

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

Newsgroup disclaimer
iAnywhere Solutions has no obligation to provide solutions, information, or ideas on its newsgroups, nor is iAnywhere Solutions obliged to provide anything other than a systems operator to monitor the service and ensure its operation and availability.
iAnywhere Solutions Technical Advisors as well as other staff assist on the newsgroup service when they have time available. They offer their help on a volunteer basis and may not be available on a regular basis to provide solutions and information. Their ability to help is based on their workload.

Feedback

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

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

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

Part I. Using MobiLink Technology

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

CHAPTER 1

Introducing MobiLink Synchronization

Contents

About this chapter

This chapter introduces you to MobiLink synchronization technology. It describes the purpose and characteristics of MobiLink.

☞ For information about MobiLink clients, see “Introducing MobiLink Clients” [*MobiLink - Client Administration*].

☞ For a more detailed introduction to MobiLink technology, see “Synchronization Basics” on page 21.

The MobiLink synchronization process

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 (SQL Anywhere, Adaptive Server Enterprise, Oracle, Microsoft SQL Server, or IBM DB2). 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.

MobiLink features

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

Features

- **Easy to get started** Using the Create Synchronization Model wizard, you can create synchronization applications very quickly. The wizard can handle many difficult implementation details of complex synchronization systems. MobiLink Design Mode in 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 two mechanisms for monitoring your synchronizations: the MobiLink Monitor and statistical scripts.
- **Performance tuning** There are a number of 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.

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** You can choose to perform only an upload or only a download, as well as doing a two-way synchronization.
- **File-based download** Downloads can be distributed as files, enabling offline distribution of synchronization changes. This 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, as well as cause remote databases to upload data to the consolidated database.
- **Choice of network protocols** Synchronization can be carried out over TCP/IP, HTTP, or HTTPS. Palm devices can synchronize through HotSync. Windows CE devices can synchronize using 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. (You can also choose to have each transaction on the remote uploaded as a separate transaction if you want to preserve the order of transactions.)

  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 (SQL Anywhere, Adaptive Server Enterprise, Oracle, Microsoft SQL Server, or IBM DB2 UDB), 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 or UNIX/Linux platforms. SQL Anywhere runs on Windows, Windows CE, or UNIX. UltraLite runs on Palm or Windows CE.

Sample applications

MobiLink samples are installed with the product in *samples-dir\MobiLink*. For more information about *samples-dir*, see “The samples directory” [*SQL Anywhere Server - Database Administration*].

MobiLink code exchange samples are located at http://ianywhere.codexchange.sybase.com/servlets/ProjectDocumentList?folderID=319.

CHAPTER 2

MobiLink Plug-in for Sybase Central

Contents

About this chapter

This chapter describes how to set up and maintain MobiLink applications using Sybase Central.

Creating and configuring a model

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. When you run the wizard, you are prompted to connect to a consolidated database in order to obtain schema information. If your database is not yet set up as a consolidated database, the wizard can apply setup scripts to create MobiLink system tables and other objects required by synchronization; other than that, no changes are made to your consolidated database until you deploy your model.

After you complete the Create Synchronization Model wizard, the model appears in **Model mode**. You can use Model mode to further customize your model. When you are in Model mode, you are working offline: no changes are made to your consolidated database. Your model is stored in a model file with the extension *.mlsm*.

When your model is complete, you use the **Deploy Synchronization Model wizard** to deploy it. The Deploy wizard can apply changes directly to the remote and consolidated databases or it can create files that you can run later. It can also create script files to run the synchronization server and client, depending on the deployment options you choose.

Sybase Central's model creation functionality includes an **Update Schema wizard** to let you update the consolidated or remote databases in the model, if required. For example, if you deploy a model that adds a timestamp column to each synchronized table on the consolidated database, you need to update the consolidated schema in the model before redeploying.

Sybase Central provides functionality to customize your model within Model mode. If you customize the model outside of Model mode, you cannot reverse-engineer a MobiLink application in your consolidated database back into the *.mlsm* file.

Create Synchronization Model wizard

The Create Synchronization Model wizard takes you through all the steps of creating a MobiLink synchronization application.

♦ Overview of setting up a MobiLink application with the Create Synchronization Model wizard

1. Open Sybase Central.

   Choose Programs ► SQL Anywhere 10 ► Sybase Central.

2. In the right pane, double-click MobiLink 10.

3. Choose File ► New ► Synchronization Model.

   The Create Synchronization Model wizard appears.

4. Choose a name and location for your model.

   Note: This information is used as defaults for files and folders that are created by the wizard.

5. The Primary Key Requirements page opens. This page is included for safety. It reminds you to maintain unique, permanent primary keys. To continue, you must select the three checkboxes. You can disable this page for the future by selecting the box at the bottom of the page.

6. The Consolidated Database Schema page opens. You must connect to the database that will be the consolidated database in your MobiLink application so that the wizard can obtain schema information for it. If this database has not been set up for use as a consolidated database, the wizard prompts you to do so.

   The MobiLink system setup process adds system objects that are required by MobiLink to the database. See "Setting up a consolidated database" [*MobiLink - Server Administration*]. These changes are made to the consolidated database immediately. (Other changes are not made until you deploy your model using the Deploy Synchronization Model wizard.)

   The connection to the database is closed if you connect to a different database, or when you exit the wizard.

7. Follow the remaining instructions in the wizard. Default recommendations based on best practices are used where possible. There is online help for all the pages. Changes are not made to the consolidated or remote databases until you deploy the model.

8. Click Finish.

   When you click Finish, the model you just created opens in Sybase Central Model mode. At the same time, your connection to the consolidated database is closed. You are now working offline. 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 "Model mode" on page 9.

Model mode

Model mode allows you to customize the defaults generated by the wizard. For example, from Model mode you can customize one table to be download-only and another to be upload-only.

When you are working in Model mode, you are offline. Changes you make are made to the model file. When you deploy, you can choose whether to make changes to your consolidated database or create files that you can run later to make the changes.

Modifying table mappings and synchronization options

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

On the Mappings tab, which opens by default when you finish the Create Synchronization Model wizard or open a model, you can view and modify information about the tables you want to synchronize.

The Mappings tab has two panes: an upper pane that contains a list of all your synchronized tables, and a lower pane with several tabs that provide details about the table mappings.

Table mappings upper pane

You can mark a table mapping for deletion by right-clicking the row in the upper pane and choosing Delete. The table mapping is deleted when you save the model. Deleting a table mapping means that the table will not be synchronized.

To view or modify the scripts for a table mapping, highlight the row and choose File ► Go To Events.

The Table Mappings upper pane contains the following columns. The values in the columns show the values you chose in the Create Synchronization Model wizard, or the wizard's defaults if you did not make a choice. You can modify these values on a table-by-table basis by editing this table. The columns are:

Ordinal column

This column has no title. It numbers the gridlines in the table. If you change the sorting of the tables (by clicking column headings), the grid numbers do not change with the reorder.

Status column

This column notifies you if there is an error, warning, or informational message for the table mapping. If an icon appears in this column, open the Status tab in the lower pane to see the content of the message.

Remote Table column

This is the remote table that you are synchronizing in the remote database.

For a new SQL Anywhere remote database, the owner of the remote table is the same as the owner of the corresponding consolidated database table. If you want a different owner, you should instead use an existing remote database with your desired table ownership.

Mapping Direction column

The direction of synchronization. This can be bi-directional, upload-only, or download-only.

☞ For more information about these options, see "Upload-only and download-only synchronizations" [*MobiLink - Server Administration*].

Consolidated Table column

This is the synchronization mapping for the remote table. Synchronization will be between these tables and the tables in the Remote Table column. Look at the Column Mappings tab in the lower pane to check how the remote columns are mapped to the consolidated columns for the selected table mapping.

Download Type column

Your options in this column are:

- ♦ **Timestamp** Data is downloaded only if it changed since the last download. Timestamp-based downloads require a Timestamp column for each consolidated table to track when changes are made to rows. This can be done by adding a column to the consolidated database tables or by creating shadow tables. When you choose Timestamp, the necessary objects are generated for you. You can view and modify the implementation in the Download Type tab in the lower pane.
- ♦ **Snapshot** All data in the consolidated table is downloaded in every synchronization, even if it has already been downloaded.

- **Custom** Write your own download_cursor and download_delete_cursor scripts instead of having them generated automatically. You can do this in the Events tab.

For more information about download types, see:

- "Timestamp-based downloads" [*MobiLink - Server Administration*]
- "Snapshot synchronization" [*MobiLink - Server Administration*]
- "Writing download_cursor scripts" [*MobiLink - Server Administration*]
- "Writing download_delete_cursor scripts" [*MobiLink - Server Administration*]

Download Deletes column

If this checkbox is selected, data that is deleted on the consolidated database will be deleted on the remote databases.

If you are using timestamp-based synchronization and this column is selected, the deletions on the consolidated database need to be recorded. There are two ways of doing this: using shadow tables (the default), or using logical deletes. Open the Download Deletes tab in the lower pane to view and modify the implementation.

If you are using snapshot downloads and this column is selected, all existing rows in the remote database will be deleted before the downloaded rows are added.

This option cannot be selected if the download type is Custom. In that case, you must write your own download_delete_cursor script. You can do this on the Events tab.

Download Subset column

Your options in this column are:

- **None** Download the same data to each remote database.
- **User** Each synchronization user (also known as a MobiLink user) only downloads rows where the user name matches a column from the same table or a directly joined table. You can view or modify the implementation on the Download Subset tab in the lower pane.
- **Remote** Each remote database only downloads rows where the remote ID matches a column from the same table or a directly joined table. You can view or modify the implementation on the Download Subset tab in the lower pane.
- **Custom** Each synchronization only downloads rows when a SQL expression in the download cursor WHERE clause is true. This option is useful when you need multiple joins or particular business logic in your download cursor WHERE clause. You can view or modify this expression on the Download Subset tab in the lower pane.

Conflict Detection column

Your options in this column are:

- **None** Always apply uploaded updates without checking for conflicts. This option results in the best performance.

- **Row-based** A conflict is detected if the row has been updated on both the remote and the consolidated database since the last synchronization.
- **Column-based** A conflict is detected only if the same column has been updated for the row in both the remote and consolidated databases. Otherwise, only the uploaded column updates are applied. If a table has BLOB columns, row-based conflict detection is used instead.

☞ For more information, see "Detecting conflicts" [*MobiLink - Server Administration*].

Conflict Resolution column

Your options in this column are:

- **Consolidated** First in wins. Uploaded updates are rejected if they conflict.
- **Remote** Last in wins. Uploaded updates are always applied. Only use this option with column-based detection; otherwise, disable conflict detection for the same result with better performance.
- **Timestamp** The newer change wins, using a timestamp column that you maintain. This timestamp column records the last time that the row was changed. It should exist in both the remote and consolidated tables and be included in the synchronization. You also must ensure that the system clocks are synchronized on the remote and consolidated databases, and that you use the same time zone (preferably UTC) for the timestamp values.
- **Custom** Write your own resolve_conflict scripts instead of having them generated automatically. You can do this on the Events tab.

☞ For more information, see "Resolving conflicts" [*MobiLink - Server Administration*].

Table mappings lower pane

When you select a table mapping row in the Table Mappings upper pane, details appear in the lower pane. The lower pane has tabs, as follows.

Column Mappings tab

If your remote database existed before you created your synchronization model, the information in this tab represents a best guess that you should check.

The Column Mappings tab shows the following information for the selected table mapping:

- **Synchronize** Columns with a checkmark are synchronized.
- **Remote column name** The name of the column in the remote database table.
- **Remote column number** This tells you the order of the columns as they were created.
- **Remote column data type** The data type.
- **Consolidated column name** The name of the column in the consolidated database table.
- **Consolidated column number** This tells you the order of the columns as they were created.
- **Consolidated column data type** The data type.

Download Type tab

The Download Type tab shows the following information for the selected table:

- **Timestamp column name** The name of the column to be used for timestamp-based downloads.
- **Use timestamp columns in synchronized table** When this option is selected, a timestamp column is added to each synchronized table if it does not already exist.
- **Use shadow tables to hold timestamp columns** When this option is selected, a shadow table is created for each synchronized consolidated database table.

Download Deletes tab

The Download Deletes tab shows the following information for the selected table:

- **Use shadow tables to record deletions** Choose this option if you want to use shadow tables to track deletes in the consolidated database. The default Timestamp column name is **last_modified**.
- **Use logical deletes** Choose this option if you want to use logical deletes in the consolidated database to help with download deletes.
  - **Column name** The name of the column that is used to indicate if a row has been deleted. This columns is added if it is not already present. The default name is **deleted**.
  - **Value if row deleted** The value that is generated when a row is deleted. The default is **Y**.
  - **Value if row not deleted** The value that is generated when a row is not deleted. The default is **N**.

For more information, see:

- "Handling deletes" [*MobiLink - Server Administration*]
- "Writing download_delete_cursor scripts" [*MobiLink - Server Administration*]

Download Subset tab

If you chose to subset data by user or remote ID, the Download Subset tab shows the following information for the selected table:

- **Use a column in each downloaded table** Choose a column containing the MobiLink user name or remote ID.
- **Use a column in a joined relationship table** Alternatively, you can identify the MobiLink user name or remote ID with a join. You can specify:
  - Table to join
  - Column to match
  - Join condition

If you chose to create a custom subset, the Download Subset tab allows you to specify tables and an expression. Multiple tables should be separated by commas.

Conflict Resolution tab

The Conflict Resolution tab shows the following information for the selected table:

- **Timestamp column** If you chose to resolve conflicts based on a timestamp, you can enter the timestamp column here.

Status tab

The status tab shows any informational, warning, or error messages for the table mapping that is selected in the upper pane.

New Table Mappings wizard

To add table mappings to your model, click File ► New ► Table Mappings. The New Table Mappings wizard appears. This wizard allows you to select remote tables that are not currently being synchronized and map them for synchronization.

You can add table mappings only if there are tables in your model that are not already mapped. If there are none, you are prompted to open the Update Schema wizard.

Modifying scripts in Model mode

In MobiLink Model mode, open the Events tab to view and modify the scripts that were generated by the Create Synchronization Model wizard or to create new scripts.

If you modify a script but then want to restore it to its original state, place your cursor in the script and choose File ► Restore *event-name* Script.

Tip
Modifying your scripts in Model mode is generally preferable to modifying them in the database or generated SQL file, because if you redeploy your model your changes will not be lost.

Authenticating to external servers in Model mode

To authenticate to external POP3, IMAP, or LDAP servers, open the Authentication tab in Model mode and select Enable Custom Authentication for this Synchronization Model.

You must enter information about the host and port, or for LDAP servers, the URL of the LDAP server.

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

Setting up server-initiated synchronization in Model mode

Sybase Central Model mode provides a way for you to set up server-initiated synchronization. This method provides a limited version of server-initiated synchronization that is very easy to set up and run. You can use the SYNC gateway from this setup, but not the UDP or SMS gateways.

In this version of server-initiated synchronization, the MobiLink server uses a download_cursor script for a table to determine when to initiate a synchronization. It does this by using the download_cursor script to generate a request_cursor for the Notifier.

♦ To set up server-initiated synchronization (Sybase Central Model mode and Deploy Synchronization Model wizard)

1. Using the Create Synchronization Model wizard, create a MobiLink model.

   ☞ See “Create Synchronization Model wizard” on page 8.

2. With your model open in Sybase Central, open the Notification tab at the top of the model.

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 that triggers a synchronization.

   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 download cursor.

5. Optionally, select other options on the Notification tab.

6. Deploy your model:

   a. From the File menu, choose Deploy.

   b. Follow the instructions in the Deploy Synchronization Model wizard.

      ☞ See “Deploy Synchronization Model wizard” on page 16.

   c. On the Server-Initiated Synchronization Listener page, choose options for your Listener.

7. To use server-initiated synchronization, you must:

   a. Start the MobiLink server.

   b. Perform a first synchronization (if one has not yet been done).

   c. Start the Listener.

8. Navigate to the folder you chose when you first started the Create Synchronization Model wizard. It holds your model with the extension *.mlsm*. It also holds the following sub-folders:

   ♦ *\mlsrv*

   ♦ *\remote*

- *\consolidated*

To set up server-initiated synchronization (not in Model mode)

Sybase Central Model mode provides a way to set up a simplified version of server-initiated synchronization.

☞ For a description of the complete implementation of server-initiated synchronization, see MobiLink - Server-Initiated Synchronization [*MobiLink - Server-Initiated Synchronization*].

☞ For an overview of what is required to set up server-initiated synchronization outside of Model mode, see “Quick start” [*MobiLink - Server-Initiated Synchronization*].

Deploy Synchronization Model wizard

When you are ready to try your model, you deploy it with the Deploy Synchronization Model wizard. There are multiple things that can be deployed:

- Changes to the consolidated database.
- SQL Anywhere or UltraLite remote databases (you can choose to have a database created, or have tables added to an existing empty database, or use an existing database that already has your remote tables).
- Batch files to deploy the model.
- Batch files to run the MobiLink server and the MobiLink client.
- Server-initiated synchronization configuration.

Deploying a model also has the effect of saving your changes in the file in which the model is stored.

Note:
After deploying a model, you can alter it. The best way to alter the model is to make changes in Sybase Central Model mode and re-deploy it. You can also alter your deployed system in Sybase Central's Admin mode or by directly changing the database using system procedures or using other methods. However, when you alter a deployed model, you cannot reverse-engineer it back to the model. When you deploy a model, any files of the same name are overwritten.

Redeployment

When you deploying a model that was previously deployed, you first need to run the Update Schema wizard if the previous deployment made any changes to the schema of the remote or consolidated database. For example, if you deploy a model that adds a timestamp column to each synchronized table on the consolidated database (which is default behavior when you create a model), you need to update the consolidated schema in the model before redeploying.

♦ To deploy a model

1. From the File menu, choose Deploy.

   The Deploy Synchronization Model wizard appears.

2. Follow the instructions in the wizard. Each page has online help.
3. When you are finished, the changes you selected are deployed. If there are existing files of the same name, they are overwritten.

Synchronizing a deployed model

When you deploy a model, folders and files are optionally created in the location you chose on the first screen of the Create Synchronization Model wizard. The files and folders 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:

Folders (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.

Folders (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, 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 set up a new SQL Anywhere remote database, this is the database. ♦ *MyModel_remote.sql* - a SQL file for setting up the remote database. ♦ *MyModel_remote.udb* - if you set up a new UltraLite remote database, this is the database. ♦ *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 you must include connection information. You may need to create ODBC data sources before running these batch files.

☞ See "Working with ODBC data sources" [*SQL Anywhere Server - Database Administration*].

♦ To synchronize your model using batch files

1. In the Consolidated Database Deployment Destination page of the Deploy Synchronization Model wizard, you chose whether you wanted to deploy directly to the consolidated database or create a file to run later. If you chose the latter, you must run the batch file 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*. You must include connection information. For example, run:

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

2. In the New SQL Anywhere Remote Database page or New UltraLite Remote Database page of the Deploy Synchronization Model wizard, you chose whether you wanted to deploy directly to the remote database or create a file to run later. If you chose the latter, 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*. For example, run:

```
MyModel_remote.bat
```

3. 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 example, run:

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

4. 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:

     ```
     DBENG10 MyModel_remote.db
     ```

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

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

   For an UltraLite remote database:

   ♦ To test your synchronization, run the file that ends with *_ulsync.bat* in the *remote* folder.

   ♦ Alternatively, run your UltraLite application.

5. 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*]

Admin mode

Admin mode is legacy functionality in Sybase Central.

When you work in Admin mode, you are connected to your consolidated or remote database and working live. In Model mode, by contrast, you are in a disconnected state, making changes to an XML file that you later deploy.

Admin mode allows you to perform in a graphical interface the tasks that many users otherwise do in Model mode, or programmatically, or with system procedures or SQL statements.

When you make changes outside of Model mode, you cannot reverse-engineer them into the model.

☞ For more information about Admin mode, see the relevant sections of the documentation. You can see the complete listing of online help for Admin mode at "MobiLink Admin Mode Help" [*SQL Anywhere 10 - Context-Sensitive Help*].

CHAPTER 3

Synchronization Basics

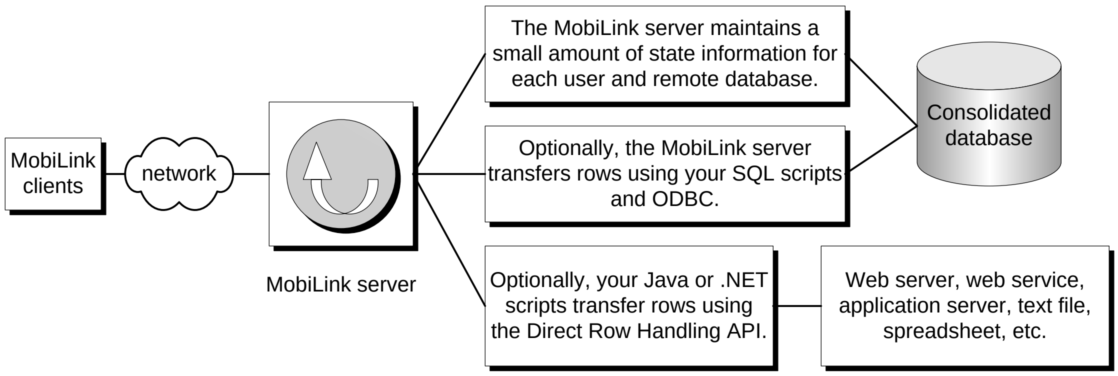
Contents

About this chapter

This chapter introduces the basic components of MobiLink technology and provides information about how to set up your synchronization system.

Parts of the synchronization system

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


MobiLink clients
network
MobiLink server
The MobiLink server maintains a small amount of state information for each user and remote database.
Optionally, the MobiLink server transfers rows using your SQL scripts and ODBC.
Optionally, your Java or .NET scripts transfer rows using the Direct Row Handling API.
Consolidated database
Web server, web service, application server, text file, spreadsheet, etc.

- **MobiLink clients** The client can be installed on a handheld device such as a Palm Pilot or Pocket PC, 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 single MobiLink installation.

  For more information, see "MobiLink clients" on page 25.

- **Network** The connection between the MobiLink server and the MobiLink client can use a number of protocols.

  For more information, see:

  - MobiLink server: "-x 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.

  For more information, see "MobiLink Server" [*MobiLink - Server Administration*].

- **Consolidated database** This database holds system tables and procedures that are required by MobiLink synchronization, as well as state information needed to synchronize. It also typically contains the central copy of information in the synchronization system. This database can be SQL Anywhere, Adaptive Server Enterprise, Oracle, DB2, or Microsoft SQL Server.

  For more information, see "Central data sources" on page 24 and "MobiLink Consolidated Databases" [*MobiLink - Server Administration*].

- **State information** The MobiLink server must maintain some information in system tables in the consolidated database. It does this over an ODBC connection.

- **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.

For more information, see "iAnywhere Solutions ODBC Drivers for MobiLink" [*MobiLink - Server Administration*].

- ♦ **Direct row handling** In addition to a consolidated database, you can optionally synchronize with other data sources using the MobiLink Direct Row Handling API. These connections could use a variety of interfaces.

  For more information, see "Direct Row Handling" [*MobiLink - Server Administration*].

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.

☞ For more information about synchronization scripts, see "Writing Synchronization Scripts" [*MobiLink - Server Administration*].

Central data sources

A MobiLink synchronization system can have two types of central data source:

- **Consolidated database** You must have a consolidated database. It can be SQL Anywhere, Adaptive Server Enterprise, Oracle, Microsoft SQL Server, or IBM DB2. (MobiLink supports DB2 for Linux, Unix, and Windows, but does not support DB2 for AS/400 or mainframe.) To prepare a database to be a consolidated database, you run a setup script that installs MobiLink system tables, stored procedures, triggers, and events that are required by synchronization. The MobiLink system tables hold state, configuration, and user information that is required for synchronization. In most applications, the consolidated database is also the master repository of information in the synchronization system.

  ☞ See “MobiLink Consolidated Databases” [*MobiLink - Server Administration*].

- **Another central data source** Optionally, you can store all or part of your central data in another data source. This can be, for example, an application server, spreadsheet, web server, web service, or text file. You can use the MobiLink Direct Row Handling API to read the upload and create the download. You can write your synchronization logic in Java or .NET scripts.

  ☞ See “Direct Row Handling” [*MobiLink - Server Administration*].

MobiLink clients

Each remote database, together with its applications, is referred to as a **MobiLink client.** Two types of MobiLink client are supported:

- SQL Anywhere
- UltraLite

☞ For more information, see "Introducing MobiLink Clients" [*MobiLink - Client Administration*].

Remote databases can contain the same tables as the central data sources, or a subset, or a different schema altogether.

☞ For information about how to handle different schemas, see "Partitioning rows among remote databases" [*MobiLink - Server Administration*].

MobiLink server

The MobiLink server, mlsrv10, sits between MobiLink clients and the central data source(s), and all communication between client and server goes through it.

For more information about how to run mlsrv10, see:

- “MobiLink Server” [*MobiLink - Server Administration*]
- “MobiLink Server Options” [*MobiLink - Server Administration*]

The synchronization process

A **synchronization** is a process of data exchange between MobiLink clients and a synchronization server. During this process, the client must establish and maintain a session with the synchronization 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 MobiLink client 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.

1. **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 synchronization 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.

   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*].

2. **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 synchronization server builds and downloads the entire download. Limiting the chattiness of the protocol is especially important when communication is slower and has higher latency, as is the case when using telephone lines or public wireless networks.

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, a number of synchronization events occur. At the occurrence of an event, MobiLink looks for a script to match the synchronization event. This 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.

For more information about synchronization events and scripts, see:

- "Writing Synchronization Scripts" [*MobiLink - Server Administration*]
- "Synchronization Events" [*MobiLink - Server Administration*]

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, in some cases you may need to edit them.

Following are the typical upload scripts for tables:

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 first event, upload_insert, triggers the running of the upload_insert script, which inserts the emp_id and emp_name into the emp table. In like fashion, the upload_delete and upload_update scripts will perform similar functions for delete and update actions on the same emp table.

The download script uses a cursor. 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}
```

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.

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 have provide classes and methods to suit the needs of synchronization.

☞ For more information about programming synchronization logic, see “Options for writing synchronization logic” on page 35.

☞ For information about DBMS-dependent scripting, such as scripting for Oracle, MS SQL Server, IBM DB2 UDB, or Adaptive Server Enterprise databases, 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.

☞ For more information, 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 and commits these changes once it has completed inserting new rows, deleting old rows, making updates, and resolving any conflicts.

The MobiLink server prepares the download, including all deletes, inserts, and updates, using another transaction. If you specify download acknowledgement and the client confirms a successful download, the MobiLink server commits the download transaction. If the application encounters problems or cannot reply when download acknowledgement is specified, the MobiLink server instead rolls back the download transaction. The default is to not use download acknowledgement.

Do not commit or roll back transactions within a script
COMMIT or ROLLBACK statements within 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. There should be no implicit or explicit commit or rollback in your synchronization scripts or the procedures or triggers that are called from your synchronization scripts.

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. Thus, 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.

Similarly, 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.

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 in case they must be sent again. 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 you specify download acknowledgement, if the failure occurs in 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 use download acknowledgement, the MobiLink server automatically rolls back the download transaction in the consolidated database.

In all cases where failure may occur, no data is lost. The MobiLink server and the MobiLink client manage this for you. The developer or user need not 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. When a row is being added to the download, the MobiLink server locates the row in the upload and omits the row from the download if it is found to be the same.

- **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 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 affords you these key 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 will remove 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.

- For SQL Anywhere clients, dbmlsync writes an entry in a 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, a warning SQLE_ROW_DELETED_TO_MAINTAIN_REFERENTIAL_INTEGRITY is raised. This warning takes a parameter, which is the table name. The warning is raised on every row that is deleted to maintain referential integrity. 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 deleted.

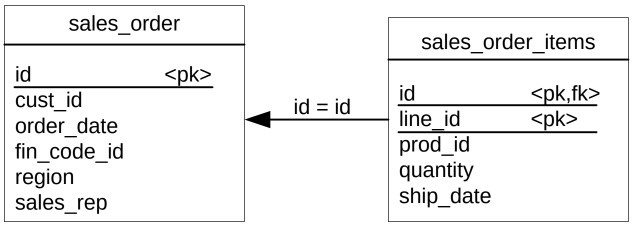
  If you want to fail synchronization when the warning is raised, you must implement a synchronization observer and then signal the observer (perhaps through a global variable) from the error callback function to have the synchronization fail on the next call to the observer.

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 as rows are inserted, updated, and deleted, but the rows that violate referential integrity are automatically removed before the download is committed.

Example

Suppose that an UltraLite sales application contains, among others, 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 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_item 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*]

Options for writing 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 (SQL Anywhere, Adaptive Server Enterprise, Oracle, SQL Server, or DB2).

Java or .NET are useful if you are synchronizing against something other than a supported consolidated database, and 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 other than a consolidated database, *does* directly manipulate the upload and download rows—without using the consolidated database.

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 MobiLink authentication integrates with your corporate security policies.
- **Stored procedures** If your database lacks the ability to make user-defined stored procedures, you can create a method in Java or .NET that can perform the needed functionality.
- **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-based row handling and direct row handling.
- The full richness of the Java language and its large body of existing code and libraries.

☞ For more information, 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-based 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-based row handling and direct row handling.

☞ For more information, see “MobiLink server API for .NET reference” [*MobiLink - Server Administration*].

Further reading

For more information about your options for writing synchronization scripts, see:

- “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*]

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 DBMS user authentication system and other security features.

  For more information, see your DBMS 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 are designed to prevent unauthorized access through client/server communications, but not to be proof against 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 "Security considerations" [*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.

  For more information, 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.

  For more information, see "MobiLink Users" [*MobiLink - Client Administration*].

Part II. MobiLink Tutorials

This part 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.

CHAPTER 4

Exploring the CustDB Sample for MobiLink

Contents

About this chapter

This chapter uses the CustDB sample application to illustrate a variety of techniques that you can use for common synchronization tasks.

Introduction

This chapter introduces you to the CustDB (Customer Database) MobiLink sample application. 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 will need to develop MobiLink applications.

The application has been designed to illustrate several common synchronization techniques. To get the most out of this chapter, you should 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 44.

Scenario

The CustDB scenario is as follows.

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.

For more information, 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.

  For more information, 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.

  For more information, 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 “About the CustDB sample database” [*SQL Anywhere 10 - Introduction*].

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 "The samples directory" [*SQL Anywhere Server - Database Administration*].

Setting up the CustDB consolidated database

The CustDB consolidated database may be SQL Anywhere, Sybase Adaptive Server Enterprise, Microsoft SQL Server, Oracle, or IBM DB2.

SQL Anywhere CustDB

A SQL Anywhere CustDB consolidated database is provided in *samples-dir\UltraLite\CustDB\custdb.db*. A DSN called SQL Anywhere 10 CustDB is included with your installation.

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

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*
Microsoft SQL Server	*custmss.sql*
Oracle	*custora.sql*
IBM DB2	*custdb2.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, Oracle, or SQL Server)

1. Create a database in your RDBMS.
2. Add the MobiLink system tables by running one of the following SQL scripts, located in the *MobiLink \setup* subdirectory of your SQL Anywhere 10 installation:
   - ♦ For an Adaptive Server Enterprise consolidated database, run *syncase.sql*.
   - ♦ For an Oracle consolidated database, run *syncora.sql*.
   - ♦ For a SQL Server consolidated database, run *syncmss.sql*.
3. Add sample user tables to the CustDB database by running one of the following SQL scripts, located in the *Samples\MobiLink\CustDB* subdirectory of your SQL Anywhere 10 installation:
   - ♦ For an Adaptive Server Enterprise consolidated database, run *custase.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 machine.
   - ♦ Choose Start ► Programs ► SQL Anywhere 10 ► SQL Anywhere ► ODBC Administrator.
   - ♦ Click Add.
   - ♦ Select the appropriate driver from the list.

     Click Finish.
   - ♦ Name the ODBC data source CustDB.
   - ♦ Click the Login tab. Enter the user ID and password for your database.

♦ To set up a consolidated database (IBM DB2)

1. Create a DB2 database on the DB2 server. For the purposes of this tutorial, call 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:
   - ♦ Verify that at least one of your buffer pools has 8 KB pages. If not, create a buffer pool with 8 KB pages.
   - ♦ Create a new table space and temporary table space with 8 KB pages.

     For more information, consult your DB2 documentation.
3. Add the MobiLink system tables to the DB2 consolidated database using the file *MobiLink\setup \syncdb2long.sql*:

- ♦ Change the connect command at the top of the file *syncdb2long.sql*. Replace *DB2Database* with the name of your DB2 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 ~
  ```

- ♦ Open a DB2 Command Window on either the server or client computer. Run *syncdb2long.sql* by typing the following command:

  ```
      db2 -c -ec -td~ +s -v -f syncdb2long.sql
  ```

4. In order for DB2 to use the stored procedures defined in *syncdb2long.sql*, you must copy the *syncdb2long_version* Java and class files located in the *MobiLink\setup* subdirectory of your SQL Anywhere installation to the *FUNCTION* subdirectory of your DB2 installation.

5. Copy *custdb2.class*, located in the *Samples\MobiLink\CustDB* subdirectory of your SQL Anywhere installation, to the *SQLLIB\FUNCTION* directory on your DB2 server machine. This class contains procedures used for the CustDB sample.

6. Add data tables to the CustDB database:

   - ♦ 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
     ```

   - ♦ Open a DB2 Command Window on either the server or client computer.

   - ♦ Run custdb2.sql by typing the following command:

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

   - ♦ When processing is complete, enter the following command to close the command window:

     ```
     exit
     ```

7. Create an ODBC data source called CustDB that references the DB2 database on the DB2 client machine.

   - ♦ Start the ODBC Administrator:

     From the Start menu, choose Programs ► SQL Anywhere 10 ► SQL Anywhere ► ODBC Administrator.

     The ODBC Data Source Administrator appears.

   - ♦ On the User DSN tab, click Add.

     The Create New Data Source dialog appears.

   - ♦ Select the ODBC driver for your DB2 database. For example, choose IBM DB2 UDB ODBC Driver. Click Finish.

     For information about how to configure your DB2 ODBC driver, see:

- ♦ Your DB2 documentation
- ♦ http://www.ianywhere.com/developer/technotes/odbc_mobilink.html

8. Run the *custdb2setuplong* Java application on the DB2 client machine as follows. This application resets the CustDB example in the DB2 database. After the initial setup, you can run this application at any time to reset the DB2 CustDB database by typing the same command line.

   - ♦ If you use a name other than CustDB for the data source, you must modify the connection code in *custdb2setuplong.java* and recompile it as follows. If the path specified by the system variable *%db2tempdir%* contains spaces, you must enclose the path in quotation marks.

     ```
     javac -g -classpath %db2tempdir%\java\jdk\lib\classes.zip;
     %db2tempdir%\java\db2java.zip;
     %db2tempdir%\java\runtime.zip custdb2setuplong.java
     ```

   - ♦ Type the following, where *userid* and *password* are the user name and password for connecting to the CustDB ODBC data source.

     ```
     java custdb2setuplong userid password
     ```

☞ For more information about DB2, see "IBM DB2 UDB 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 the sample application to your remote device.
4. Start the sample application on the remote device.

5. Synchronize the sample application.

Example

The following example installs the CustDB sample on a Palm device running against a DB2 consolidated database.

1. Ensure that the consolidated database is running:

   For a DB2 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 database:

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

2. Start the MobiLink server:

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

   ```
   mlsrv10 -c "DSN=CustDB" -zp
   ```

3. Install the sample application to your Palm device:

   ♦ On your PC, start Palm Desktop.

   ♦ Click Quick Install on the Palm Desktop toolbar.

   ♦ Click Add. Browse to *custdb.prc* in the *UltraLite\palm\68k* subdirectory of your SQL Anywhere 10 installation.

   ♦ Click Open.

   ♦ HotSync your Palm device.

4. Start the CustDB sample application on your Palm device:

   ♦ Place your Palm device in its cradle.

   When you start the sample application for the first time, you are prompted to synchronize to download an initial copy of the data. This step is required only the first time you start the application. After that, the downloaded data is stored in the UltraLite database.

   ♦ Launch the sample application.

   From the Applications view, tap CustDB.

   An initial dialog appears, prompting you for an employee ID.

   ♦ Enter an employee ID.

   For the purpose of this tutorial, enter a value of 50. The sample application also allows values of 51, 52, or 53, but behaves slightly differently in these cases.

   For more information about the behavior of each user ID, see "Users in the CustDB sample" on page 53.

A message box tells you that you must synchronize before proceeding.

- ♦ Synchronize your application.

  Use HotSync to obtain an initial copy of the data.

- ♦ Confirm that the data has been synchronized into the application.

  From the Applications view, tap the CustDB application. The display shows an entry sheet for a customer, with entries.

5. Synchronize the remote application with the consolidated database. You will only need to complete this step when you have made changes to the database.

   - ♦ Ensure that the consolidated database and the MobiLink server are running.
   - ♦ Place the Palm device in its cradle.
   - ♦ Press the HotSync button to synchronize.

Tables in the CustDB databases

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

☞ For an entity-relationship diagram of the CustDB tables, see "About the CustDB sample database" [*SQL Anywhere 10 - 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 the 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 will be 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 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 51, 52, and 53 identify remote databases that are associated with sales people. Sales people can carry out the following tasks:
  - View lists of customers and products.
  - Add new customers.
  - Add or delete orders.
  - Scroll through the list of outstanding orders.
  - Accept or deny orders.
  - Synchronize changes with the consolidated database.
- **Mobile managers** User ID 50 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 carry out the following task:
  - Accept or deny orders.

Synchronization

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 10.0',
'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 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 carried out 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 of 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 of 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 in order 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** The **download_cursor** script for CustDB is as follows:

```
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 **UL_CustomerIDPool_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 **UL_OrderIDPool_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, reset the data in the CustDB database.

♦ **To reset the data in the CustDB database**

1. Install the ULDBUtil on your device:

   ♦ For a Palm device, start Palm Desktop on your PC.

   ♦ Click Install on the Palm Desktop toolbar.

   ♦ Click Add. Browse to *uldbutil.prc* in the *UltraLite\palm\68k* subdirectory of your SQL Anywhere 10 installation.

   ♦ Click Done.

   ♦ HotSync your Palm device.

2. Delete the data using ULDBUtil:

   ♦ For a Palm device, tap the ULDBUtil icon.

   ♦ Select CustDB and tap Delete Data.

   ♦ HotSync your Palm device.

Further reading

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

☞ For reference material, including detailed information about each script and its parameters, see “Synchronization Events” [*MobiLink - Server Administration*].

CHAPTER 5

Exploring the MobiLink Contact Sample

Contents

About this chapter

This chapter uses the sample application called Contact to illustrate a variety of techniques that you can use for common synchronization tasks.

The techniques are illustrated using SQL scripts. Many of the same techniques can be implemented using Java or .NET synchronization logic.

Introduction

This chapter introduces you to the Contact sample application. This sample is a valuable resource for the MobiLink developer. It provides you with an example of how to implement many of the techniques you will 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 chapter, you should 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 more information about *samples-dir*, see “The 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 of the columns but only one of the rows of the SalesRep table on the consolidated database are shared with each remote database.

  For more information, 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.

  For more information, see “Timestamp-based downloads” [*MobiLink - Server Administration*].

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 carries out 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 more information about *samples-dir*, see "The samples directory" [*SQL Anywhere Server - Database Administration*].

Running the Contact sample

The Contact sample includes batch files that carry out 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.

   - ♦ At a command prompt, navigate to *samples-dir\MobiLink\Contact* and execute 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.

   - At a command prompt, navigate to *samples-dir\MobiLink\Contact*.
   - Execute the following command:

     ```
     step2
     ```

     This is a batch file that synchronizes both remote databases.

3. Shut down the MobiLink server.

   - At a command prompt, navigate to *samples-dir\MobiLink\Contact*.
   - Execute 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

Several different database user IDs and MobiLink user names are included in the Contact sample.

Database user IDs

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

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

Each remote database also has a user ID **sync_user** with password **sync_user**. This user ID is employed only on the dbmlsync command line. It is a user with REMOTE DBA authority, and so can carry out any operation when connected from dbmlsync, but has no authority when connected from any other application. The widespread availability of the user ID and password is thus not a problem.

At the consolidated database, there is a user named **mlmaint**, who owns the tables used for monitoring MobiLink synchronization statistics and errors. This 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.

Synchronization

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.

☞ For more information, 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.

☞ For more information, see "Timestamp-based downloads" [*MobiLink - Server Administration*].

☞ For more information, see "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 carries out 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 carried out 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 so that 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 contact 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 carried out 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 will change the quantity from 90 to 70. If Pamela Savarino takes an order for 10 baseball hats before receiving this change, she will change 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.

CHAPTER 6

Tutorial: Using MobiLink with an Oracle 10g Consolidated Database

Contents

About this chapter

This chapter provides a tutorial to guide you through the process of setting up a synchronization system when the consolidated database is an Oracle database and the remote database is a SQL Anywhere database.

Introduction

In this tutorial, you prepare an Oracle consolidated database and a SQL Anywhere remote database.

Required software

- A full SQL Anywhere installation.
- A full installation of Oracle Enterprise Edition 10g.
- iAnywhere Solutions - Oracle Wire Protocol driver, available by download. See http://www.ianywhere.com/developer/technotes/odbc_mobilink.html.

Competencies and experience

You should have the following competencies and experience before beginning the tutorial:

- Familiar with Sybase Central interface and functionality.
- Competent with Interactive SQL and Oracle SQL Plus.
- Competent programming Oracle.

Goals

The goals for the tutorial are:

- To acquire familiarity with the MobiLink server and related components as they can be used with Oracle.
- To gain competence in executing MobiLink server and client commands as they pertain to an Oracle consolidated database.

In particular, you will carry out the following tasks:

- Create a new SQL Anywhere database to serve as a remote database.
- Start a MobiLink server to work with your consolidated Oracle database.
- Start the MobiLink synchronization client and synchronized the remote database with the consolidated Oracle database.

Suggested background reading

☞ For more information about running the MobiLink server, see "Synchronization Basics" on page 21.

Lesson 1: Create your databases

MobiLink synchronization requires that you have a consolidated database (Oracle in this tutorial), a remote database (SQL Anywhere in this tutorial), and an ODBC data source for each of these databases.

Create the SQL Anywhere remote database

♦ To create your remote database

1. Create a directory for this tutorial (for example, *OracleTut*). Open a command prompt and navigate to *OracleTut*.
2. Type the following command:

   ```
   dbinit remote.db
   ```

3. Verify the successful creation of the database by listing the contents of *OracleTut*.

♦ To create an ODBC data source for the remote database

- While still at your *OracleTut* directory, type the following command on a single line:

  ```
  dbdsn -w test_remote -y
     -c "uid=DBA;pwd=sql;dbf=path\OracleTut\remote.db;eng=remote"
  ```

  Replace *path* with the location of your *OracleTut* directory.

♦ To create objects in the remote database

1. Start Interactive SQL.

   Choose Start ► Programs ► SQL Anywhere 10 ► Interactive SQL.

2. Connect to the remote database.
3. Create remote tables, a publication, a user, and a subscription:

   Copy the following code into Interactive SQL and execute it.

   ```
   CREATE TABLE emp ( emp_id int primary key ,emp_name varchar( 128 ) );
   CREATE TABLE cust(
      cust_id int primary key,
      emp_id int references emp ( emp_id ),
      cust_name varchar( 128 ) );
   CREATE PUBLICATION emp_cust ( TABLE cust, TABLE emp );
   CREATE SYNCHRONIZATION USER ml_user;
   CREATE SYNCHRONIZATION SUBSCRIPTION
     TO emp_cust FOR ml_user TYPE TCPIP ADDRESS 'host=localhost';
   ```

Further reading

☞ For more information about creating SQL Anywhere databases, see "The Initialization utility" [*SQL Anywhere Server - Database Administration*].

☞ For more information about creating an ODBC data source for a SQL Anywhere database, see "The Data Source utility" [*SQL Anywhere Server - Database Administration*].

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

For information about creating the SQL Anywhere objects in this tutorial, see:

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

Create the Oracle consolidated database

You can enter data into an Oracle database using a number of different methods. This tutorial uses Oracle SQL Plus.

♦ Create your Oracle consolidated database

1. Start SQL Plus.

   Choose Start ► Programs ► Oracle - OraDb10g_home1 ► Application Development ► SQL Plus.

2. Connect to your consolidated database.

3. Copy the following code into SQL Plus and execute it. These SQL statements drop, create and populate tables in the consolidated database. If there are no tables to drop, an error appears in the SQL Plus output, but this will not affect processing.

```
CREATE SEQUENCE emp_sequence;
CREATE SEQUENCE cust_sequence;
DROP TABLE emp;
CREATE TABLE emp(
   emp_id int primary key,
   emp_name varchar( 128 ) );
DROP TABLE cust;
CREATE TABLE cust(
   cust_id int primary key,
   emp_id int references emp(emp_id),
   cust_name varchar( 128 ) );
INSERT INTO emp
   ( emp_id, emp_name ) VALUES ( emp_sequence.nextval, 'emp1' );
INSERT INTO emp
   ( emp_id, emp_name ) VALUES ( emp_sequence.nextval, 'emp2' );
INSERT INTO emp
   ( emp_id, emp_name ) VALUES ( emp_sequence.nextval, 'emp3' );
COMMIT;
INSERT INTO cust
   ( cust_id, emp_id, cust_name ) VALUES ( cust_sequence.nextval, 1,
'cust1' );
```

```
INSERT INTO cust
   ( cust_id, emp_id, cust_name ) VALUES ( cust_sequence.nextval, 1,
'cust2' );
INSERT INTO cust
   ( cust_id, emp_id, cust_name ) VALUES ( cust_sequence.nextval, 2,
'cust3' );
COMMIT;
```

4. To verify the successful creation of the tables, run the following SQL statement for each table:

   ```
   SELECT * FROM emp;
   SELECT * FROM cust;
   ```

   Leave the consolidated database running.

Create an ODBC data source for the consolidated database

MobiLink requires an ODBC data source to perform data synchronization. For version 10.0.0, the Oracle ODBC driver must be downloaded. For information, see http://www.ianywhere.com/developer/technotes/odbc_mobilink.html.

Ensure that you know your Instance, Service, and Database names, as these values are required for the ODBC portion of the installation. These values are established at the time of your Oracle installation.

The following steps set up an ODBC configuration for the Oracle consolidated database.

♦ To set up an ODBC data source for Oracle

1. Choose Start ▶ Programs ▶ SQL Anywhere 10 ▶ SQL Anywhere ▶ ODBC Administrator.

   The ODBC Data Source Administrator opens.

2. Click Add on the User DSN tab. The Create New Data Source window appears.

3. Select **iAnywhere Solutions 10 - Oracle Wire Protocol**, and click Finish.

   The ODBC Oracle Driver Setup window appears.

4. Click the General tab and type the data source name **ora_consol**. This is the DSN used for connecting to your Oracle database. You will need it later.

5. Enter the server name. This value depends on your Oracle installation. If the server is on your computer, you may be able to leave this field blank.

6. Click the Advanced tab, and then enter a Default User Name. For this tutorial you can use **system**, or any user name with sufficient rights to create objects.

7. Click OK.

8. Click OK to close the ODBC Data Source Administrator.

Further reading

☞ For more information about Oracle, see your Oracle documentation.

☞ For more information about the Oracle ODBC driver, see the PDF that is downloaded with the driver: *iAnywhere Solutions - Oracle Wire Protocol Driver*.

Set up your consolidated database

MobiLink comes with a script called *syncora.sql*, located in the *MobiLink\setup* subdirectory of your SQL Anywhere installation. You run this script to set up your Oracle database to work with MobiLink.

Syncora.sql contains SQL statements, written in Oracle SQL, to prepare Oracle databases for use as consolidated databases. It creates a series of MobiLink system tables, triggers, and procedures for use by MobiLink. The system tables are prefaced with ML_. MobiLink uses these tables during the synchronization process.

♦ Create MobiLink system tables within Oracle

1. Start SQL Plus. Choose
   Start ▶ Programs ▶ Oracle - OraDb10g_home1 ▶ Application Development ▶ SQL Plus.

   Connect to your Oracle database using Oracle SQL Plus. Log on using the **system** schema with password **manager**.

2. Run *syncora.sql* by typing the following command:

   ```
   @path\syncora.sql;
   ```

   where *path* is the *MobiLink\setup* subdirectory of your SQL Anywhere 10 installation. If there are spaces in your path, you should enclose the path and file name in quotation marks.

♦ To verify that the system tables are installed

1. Start SQL Plus. Choose
   Start ▶ Programs ▶ Oracle - OraDb10g_home1 ▶ Application Development ▶ SQL Plus.

2. Run the following SQL statement to yield a listing of the MobiLink system tables, procedures, and triggers:

   ```
   SELECT object_name
   FROM all_objects
   WHERE object_name
   LIKE 'ML_%';
   ```

   If there are no objects starting with ML_, the procedure you just completed was not successful. In this case, you need to review the MobiLink error messages to see what went wrong; correct the problem; and then drop the MobiLink system tables as follows.

♦ To drop the MobiLink system tables (if necessary)

1. Run the following SQL statement in SQL Plus:

   ```
   select 'drop ' || object_type || ' ' || object_name || ';'
   from all_objects
   where object_name like 'ML_%';
   ```

   This statement generates a list of tables, procedures and triggers to be dropped.

2. Copy this list to a text file and save it as *drop.sql* in your *OracleTut* directory. Remove any lines that do not contain drop statements.

3. Execute the SQL statements in *drop.sql* by running the following command:

   ```
   @path\OracleTut\drop.sql;
   ```

   Replace *path* with the location of your *OracleTut* directory. Run *drop.sql* a second time to delete tables that were not removed the first time because of dependencies.

You can now repeat the instructions in Create MobiLink system tables within Oracle, above.

☞ For more information about setting up an Oracle consolidated database, see "Oracle consolidated database" [*MobiLink - Server Administration*].

Lesson 2: Start the MobiLink server

The MobiLink server can now be started from a command prompt. Since the MobiLink server is a client to the consolidated database, your consolidated database must be started prior to starting the MobiLink server.

♦ To start the MobiLink server

1. Ensure that your consolidated database is running.
2. At a command prompt, navigate to your *OracleTut* directory.
3. Start the MobiLink server.

   Type the following command:

   ```
   mlsrv10 -c "dsn=ora_consol;pwd=manager;uid=system" -o mlsrv.mls -v+ -zu+
   ```

This command line specifies the following mlsrv10 options:

- **-c** Specifies connection parameters. Note that the example above only specifies the password as the DSN contains the user ID. For more information, see "-c option" [*MobiLink - Server Administration*].
- **-o** Specifies the message log file. For more information, see "-o option" [*MobiLink - Server Administration*].
- **-v+** Sets verbose logging on. For more information, see "-v option" [*MobiLink - Server Administration*].
- **-dl** Sets the display log feature ON. For more information, see "-dl option" [*MobiLink - Server Administration*].
- **-zu+** Automates the user authentication process. For more information, see "-zu option" [*MobiLink - Server Administration*].

Further reading

☞ For more information about mlsrv10, see "MobiLink Server" [*MobiLink - Server Administration*] and "mlsrv10 syntax" [*MobiLink - Server Administration*].

Lesson 3: Start the MobiLink synchronization client

The MobiLink client may now be started from a command prompt. The MobiLink client initiates synchronization.

You can specify connection parameters for the remote database on the dbmlsync command line using the -c option.

♦ To start the MobiLink client

1. Ensure that the MobiLink server is started.
2. At a command prompt, navigate to your *OracleTut* directory.
3. Type the following command:

   ```
   dbmlsync -c "dsn=test_remote" -o dbmlsync.out -v+ -e "SendColumnNames=ON"
   ```

   This command line specifies the following dbmlsync options:

   - **-c** Supply database connection parameters. For more information, see "-c option" [*MobiLink - Client Administration*].
   - **-o** Specify the message log file. For more information, see "-o option" [*MobiLink - Client Administration*].
   - **-v+** Verbose operation. For more information, see "-v option" [*MobiLink - Client Administration*].
   - **-e** Extended options. Specifying "SendColumnNames=ON" sends column names to MobiLink. For more information, see "MobiLink SQL Anywhere Client Extended Options" [*MobiLink - Client Administration*].

Further reading

☞ For more information about dbmlsync, see "dbmlsync syntax" [*MobiLink - Client Administration*].

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*].

CHAPTER 7

Tutorial: Using Java Synchronization Logic

Contents

About this chapter

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 (mlsrv10) with an option to set the path of compiled Java classes.

Introduction

MobiLink connection-level and table-level event scripts can be written in SQL, Java or .NET. Java and .NET encapsulate event logic in class methods. In this tutorial you write Java class methods for MobiLink table-level events.

Required software

- ♦ SQL Anywhere 10.0
- ♦ Java Software Development Kit

Competencies and experience

You will require:

- ♦ familiarity with Java
- ♦ basic knowledge of MobiLink event scripts

Goals

You will gain competence and familiarity with:

- ♦ using simple Java class methods for MobiLink table-level events

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 (mlsrv10) with the -sl java option
- ♦ testing synchronization with a sample Windows client application

Suggested background reading

☞ For more information about synchronization scripts, see "Introduction to synchronization scripts" [*MobiLink - Server Administration*].

Lesson 1: Compile the CustdbScripts Java class

Java classes encapsulate synchronization logic in methods.

In this lesson, you will 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 called SQL Anywhere 10 CustDB.

CustdbScripts class

In this section, you create a Java class called CustdbScripts with logic to handle the ULCustomer upload_insert and upload_update events. You enter the CustdbScripts code in a text editor and save the file as *CustdbScripts.java*.

♦ To create CustdbScripts.java

1. Create a directory for the Java class and assembly.

   This tutorial assumes the path *c:\mljava*.

2. Using a text editor, enter the CustdbScripts code:

```
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:
   The class and associated methods must be set as public.

3. Save the file as *CustdbScripts.java* in *c:\mljava*.

Compiling 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 (Windows)**

1. At a command prompt, navigate to the folder containing *CustdbScripts.java* (*c:\mljava*).
2. Type the following command. Replace *install-dir* with the location of your SQL Anywhere installation.

```
javac custdbscripts.java -classpath "install-dir\java\mlscript.jar"
```

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*].

☞ For more information about the CustDB sample database, and using alternate RDBMS servers, see "Setting up the CustDB consolidated database" on page 44.

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 upload_update events, respectively.

In this section, you specify class methods for table-level events using two approaches:

- Using the MobiLink Admin mode in Sybase Central:

  Connect to the CustDB database with Sybase Central, change the language for the upload_insert script to Java, and specify CustdbScripts.UploadInsert to handle the event.
- Using the ml_add_java_table_script stored procedure:

  Connect to the CustDB database with Interactive SQL and execute ml_add_java_table_script, specifying CustdbScripts.DownloadCursor to handle the download_cursor event.

♦ To specify CustdbScripts.UploadInsert to handle the ULCustomer upload_insert event

1. Connect to the sample database using the Sybase Central MobiLink Admin mode:
   - Start Sybase Central.
   - Click the View menu and ensure that Folders is selected.
   - From the Connections menu, choose Connect with MobiLink 10.
   - On the Identification tab, choose the ODBC Data Source name SQL Anywhere 10 CustDB.
   - Click OK to Connect.
   - Sybase Central should now display the CustDB data source under the MobiLink 10 plug-in.
2. Delete the existing upload_insert event for the ULCustomer table:
   - In the left pane, open the Synchronized Tables folder and select the ULCustomer table. A list of table-level scripts appears in the right pane.
   - Click the table script associated with the custdb 10.0 upload_insert event. From the Edit menu, choose Delete. You will need to confirm that you want to delete the object.
3. Create a new upload_insert event for the ULCustomer table:
   - With the ULCustomer table selected in the Synchronized Tables folder, choose File ► New ► Table Script.
   - Select upload_insert as the event to create and click Next.
   - Choose to create a new script definition using the Java language.
   - Click Finish.

4. Double-click the upload_insert script to open it. Replace the contents of the script with the fully qualified method name **CustdbScripts.UploadInsert** and save the script. From the file menu, choose Save.

5. Exit Sybase Central.

This step used Sybase Central to specify a Java method as the script for the ULCustomer upload_insert event.

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 Java methods to handle synchronization events.

☞ For more information, see "ml_add_java_connection_script" [*MobiLink - Server Administration*] and "ml_add_java_table_script" [*MobiLink - Server Administration*].

♦ To specify CustdbScripts.DownloadCursor() to handle the ULCustomer download_cursor event

1. Connect to the sample database with Interactive SQL.

    - Open Interactive SQL.

      The Connect dialog appears.

    - On the Identification tab, choose the ODBC Data Source SQL Anywhere 10 CustDB.

    - On the Database tab, ensure that the following option is not selected: Search Network for Database Servers.

    - Click OK to connect.

2. Execute the following command in Interactive SQL:

```
CALL ml_add_java_table_script(
'custdb 10.0',
'ULCustomer',
'download_cursor',
'CustdbScripts.DownloadCursor');
COMMIT;
```

   Following is a description of each parameter:

Parameter	Description
custdb 10.0	The script version.
ULCustomer	The synchronized table.
download_cursor	The event name.
CustdbScripts.DownloadCursor	The fully qualified Java method.

3. Exit Interactive SQL.

In this lesson, you specified your 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 scripts with stored procedures, see "ml_add_java_connection_script" [*MobiLink - Server Administration*] and "ml_add_java_table_script" [*MobiLink - Server Administration*].

☞ For general information about adding and deleting synchronization scripts, see "Adding and deleting scripts" [*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 Virtual Machine to load on server startup.

♦ To start the MobiLink server (mlsrv10) and load Java assemblies

- At a command prompt, type following command on a single line:

  ```
  mlsrv10 -c "dsn=SQL Anywhere 10 CustDB" -sl java (-cp c:\mljava)
  ```

  A message appears that the server is started. Now the Java method is executed when the ULCustomer table upload_insert event triggers during synchronization.

Further reading

☞ For more information, see “-sl java 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. It is a simple sales-status application that you can run against the CustDB consolidated database you started in the previous lesson.

Start the application (Windows)

♦ To start and synchronize the sample application

1. Launch the sample application.

   From the Start menu, choose Programs ▶ SQL Anywhere 10 ▶ UltraLite ▶ Windows Sample Application.

2. Enter an employee ID.

   Type a value of **50** and press Enter.

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 (Windows)

♦ To add an order

1. From the Order menu, choose New.

   The Add New Order screen appears.

2. Enter a new Customer Name.

   For example, enter **Frank Javac**.

3. Choose a product, and enter the quantity and discount.
4. Press Enter 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 window appears indicating one Insert successfully uploaded to the consolidated database.

Further reading

☞ For more information about the CustDB Windows Application, see “Exploring the CustDB Sample for MobiLink” on page 41.

Cleanup

Remove tutorial materials from your computer.

♦ To remove tutorial materials from your computer

1. Return the ULCustomer table upload_insert and download_cursor scripts to their original SQL logic.

   ♦ Open Interactive SQL.

   The Connect dialog appears.

   ♦ On the Identification tab, choose the ODBC Data Source SQL Anywhere 10 CustDB.

   ♦ Click OK to Connect.

   ♦ Execute the following commands in Interactive SQL:

   ```
   CALL ml_add_table_script( 'custdb 10.0',
      'ULCustomer',
      'upload_insert',
      'INSERT INTO ULCustomer( cust_id, cust_name ) VALUES( ?, ? )' );

   CALL ml_add_table_script( 'custdb 10.0',
      'ULCustomer',
      'download_cursor',
      'SELECT "cust_id", "cust_name"
       FROM "ULCustomer" WHERE "last_modified" >= ?' );
   COMMIT;
   ```

2. Close the SQL Anywhere, MobiLink, and synchronization client windows by right-clicking each taskbar item and choosing Close.

3. Delete all tutorial-related Java sources.

   Delete the folder containing your *CustdbScripts.java* and *CustdbScripts.class* files (*c:\mljava*).

Note:
Ensure that you do not have other important files in *c:\mljava*.

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*].

CHAPTER 8

Tutorial: Using .NET Synchronization Logic

Contents

About this chapter

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 (mlsrv10) with an option that sets the path of .NET assemblies.

Introduction

MobiLink connection-level and table-level event scripts can be written in SQL, Java, or .NET. Both Java and .NET encapsulate event logic in class methods. In this tutorial you will subscribe .NET class methods to MobiLink table-level events.

Required software

- ♦ SQL Anywhere 10.0
- ♦ Microsoft .NET Framework SDK

Competencies and experience

You will require:

- ♦ familiarity with .NET
- ♦ basic knowledge of MobiLink event scripts

Goals

You will gain competence and familiarity with:

- ♦ utilizing .NET class methods for MobiLink table-level event scripts

Key concepts

This section uses the following steps to implement basic .NET synchronization using the MobiLink CustDB sample database:

- ♦ compiling the *CustdbScripts.dll* private assembly with MobiLink references
- ♦ specifying class methods for table-level events
- ♦ running the MobiLink server (mlsrv10) with the -sl dnet option
- ♦ testing synchronization with a sample Windows client application

Suggested background reading

☞ For more information about synchronization scripts, see “Introduction to 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 will 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 to drive synchronization. The CustDB ULCustomer table, for example, is a synchronized table supporting 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 called SQL Anywhere 10 CustDB.

In this section, you create a .NET class called CustdbScripts with logic to handle the ULCustomer upload_insert and upload_update events.

The CustdbScripts assembly

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 API for .NET reference” [*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 .NET or at a command prompt.

- ♦ In Visual Studio .NET, create a new Class Library and enter the CustdbScripts code. Link *iAnywhere.MobiLink.Script.dll*, and build the assembly for your class.
- ♦ At a command prompt , enter the CustdbScripts code in a text editor and save the file as *CustdbScripts.cs* (*CustdbScripts.vb* for Visual Basic .NET). 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 .NET

1. Start a new Visual C# or Visual Basic .NET Class Library project.

   Use CustdbScripts for the project Name and enter an appropriate path. This tutorial assumes the path *c:\mldnet*.

2. Enter the CustdbScripts code.

   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 .NET, 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. Add a reference to the MobiLink server API.

   ♦ From the Visual Studio .NET Project menu, choose Add Existing Item...

   ♦ Select *iAnywhere.MobiLink.Script.dll* in the *Assembly\v1* subdirectory of your SQL Anywhere installation. In Visual Studio .NET, from the Open drop down menu choose Link File. In Visual Studio 2005, from the Add menu choose Add Link.

4. Right-click the CustdbScripts project and select the table Common Properties ▶ General. Make sure that the text box called Root Namespace is cleared of all text.

5. 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.

   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 .NET, 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 .NET) in *c:\mldnet*.

4. Compile the file using the following command.

   For C#, type:

```
csc /out:c:\mldnet\custdbscripts.dll /target:library /reference:"%
sqlany10%\Assembly\v1\iAnywhere.MobiLink.Script.dll" c:\mldnet
\CustdbScripts.cs
```

   For Visual Basic .NET, type:

```
vbc /out:c:\mldnet\custdbscripts.dll /target:library /reference:"%
sqlany10%\Assembly\v1\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 API for .NET reference" [*MobiLink - Server Administration*].

☞ For more information about .NET methods, see "Methods" [*MobiLink - Server Administration*].

Lesson 2: Specify class methods for events

☞ For more information about the CustDB sample database, and using alternate RDBMS servers, see "Setting up the CustDB consolidated database" on page 44.

CustdbScripts.dll, created in the previous lesson, encapsulates the methods UploadInsert() and DownloadCursor(). These methods contain implementation for the ULCustomer upload_insert and upload_update events, respectively.

In this section, you specify class methods for table-level events using two approaches:

1. Using the MobiLink Synchronization plug-in.

   You connect to the CustDB database with Sybase Central, change the language for the upload_insert script to .NET, and specify MLExample.CustdbScripts.UploadInsert to handle the event.

2. Using the ml_add_dnet_table_script stored procedure.

   You will connect to the CustDB database with Interactive SQL and execute ml_add_dnet_table_script, specifying MLExample.CustdbScripts.DownloadCursor for the download_cursor event.

♦ To subscribe CustdbScripts.uploadInsert() to the upload_insert event for the ULCustomer table

1. Connect to the sample database using the MobiLink Synchronization plug-in:

   ♦ Start Sybase Central.

   ♦ In the View menu, ensure that Folders is selected.

   ♦ From the Connections menu, select Connect with MobiLink 10.

   ♦ On the Identification tab, choose the ODBC Data Source name SQL Anywhere 10 CustDB.

   ♦ Click OK to Connect.

   ♦ Sybase Central should now display the CustDB data source under the MobiLink 10 plug-in.

2. Change the language for the ULCustomer table upload_insert event to .NET:

   ♦ In the left pane, open the Synchronized Tables folder and select the ULCustomer table. A list of table-level scripts appears in the right pane.

   ♦ Click the table script associated with the custdb 10.0 upload_insert event. From the File menu, choose Language, and change the language to .NET.

3. Enter the fully-qualified .NET method name for the upload_insert script.

   ♦ Double-click the table script associated with the upload_insert event.

   A window revealing the script contents appears.

- Change the script contents to the fully-qualified method name, **MLExample.CustdbScripts.UploadInsert**.

> **Note:**
> The fully qualified method name is case sensitive.

- To save the script, choose Save from the File menu.

4. Exit Sybase Central.

This step used Sybase Central to specify a .NET method as the script for the ULCustomer upload_insert event.

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.

☞ For more information, see “ml_add_dnet_connection_script” [*MobiLink - Server Administration*] and “ml_add_dnet_table_script” [*MobiLink - Server Administration*].

In the next section 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.

   - Start Interactive SQL:

     Choose Start ▶ Programs ▶ SQL Anywhere 10 ▶ Interactive SQL, or type the following command at a command prompt:

     ```
     dbisql
     ```

     The Connect dialog appears.

   - On the Identification tab, choose the ODBC Data Source SQL Anywhere 10 CustDB.
   - On the Database tab, ensure the option to search for network database servers is not selected.
   - Click OK to connect.

2. Execute the following command in Interactive SQL:

   ```
   CALL ml_add_dnet_table_script(
   'custdb 10.0',
   'ULCustomer',
   'download_cursor',
   'MLExample.CustdbScripts.DownloadCursor');
   COMMIT;
   ```

   Following is a description of each parameter:

Parameter	Description
custdb 10.0	The script version.
ULCustomer	The synchronized table.
download_cursor	The event name.
MLExample,CustdbScripts.DownloadCursor	The fully qualified .NET method.

3. Exit 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 the scripts used in this lesson, see "ml_add_dnet_connection_script" [*MobiLink - Server Administration*] and "ml_add_dnet_table_script" [*MobiLink - Server Administration*].

Lesson 3: Run MobiLink 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 .NET, 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 (mlsrv10) and load .NET assemblies

- Start the MobiLink server with the -sl dnet option.

  If you used Visual Studio .NET to compile your assembly:

  At a command prompt, type the following command on a single line:

  ```
  mlsrv10 -c "dsn=SQL Anywhere 10 CustDB" -dl -o cons1.txt -v+ -sl dnet(-
  MLAutoLoadPath=c:\mldnet\CustdbScripts\CustdbScripts\bin\Debug)
  ```

  If you compiled your assembly at a command prompt:

  At a command prompt, type the following command on a single line:

  ```
  mlsrv10 -c "dsn=SQL Anywhere 10 CustDB" -dl -o cons1.txt -v+ -sl dnet(-
  MLAutoLoadPath=c:\mldnet)
  ```

  A message dialog appears indicating that the server is ready to handle requests. Now the .NET method is executed when the upload_insert events triggers during synchronization.

Further reading

☞ For more information see, “-sl dnet 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. It is a simple sales-status application that you can run against the CustDB consolidated database you started in the previous lesson.

Start the application (Windows)

♦ To start and synchronize the sample application

1. Launch the sample application.

   From the Start menu, choose Programs ► SQL Anywhere 10 ► UltraLite ► Windows Sample Application.

2. Enter an employee ID.

   Input a value of **50** and press Enter.

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

In the next section you will enter a new customer name and order details. During a subsequent synchronization, this information will be uploaded to the CustDB consolidated database and the upload_insert and download_cursor events for the ULCustomer table will trigger.

Add an order (Windows)

♦ To add an order

1. From the Order menu, choose New.

   The Add New Order dialog appears.

2. Enter a new Customer Name.

   For example, enter Frank DotNET.

3. Choose a product, and enter the quantity and discount.

4. Press Enter 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 window appears indicating that one Insert successfully uploaded to the consolidated database.

Further reading

☞ For more information about the CustDB Windows Application, see “Exploring the CustDB Sample for MobiLink” on page 41.

Cleanup

Remove tutorial materials from your computer.

♦ To remove tutorial materials from your computer

1. Return the ULCustomer table upload_insert and download_cursor scripts to their original SQL logic.

   ♦ Open Interactive SQL.

   Choose Start ▶ Programs ▶ SQL Anywhere 10 ▶ Interactive SQL, or type the following command at a command prompt:

   ```
   dbisql
   ```

   The Connect dialog appears.

   ♦ On the Identification tab, choose the ODBC Data Source SQL Anywhere 10 CustDB.

   ♦ Click OK to Connect.

   ♦ Execute the following commands in Interactive SQL:

   ```
   CALL ml_add_table_script( 'custdb 10.0',
      'ULCustomer',
      'upload_insert',
      'INSERT INTO ULCustomer( cust_id, cust_name ) VALUES( ?, ? )' );

   CALL ml_add_table_script( 'custdb 10.0',
      'ULCustomer',
      'download_cursor',
      'SELECT "cust_id", "cust_name"
       FROM "ULCustomer"
       WHERE "last_modified" >= ?' );
   ```

2. Close the SQL Anywhere, MobiLink, and synchronization client windows by right-clicking each taskbar item and choosing Close.

3. Delete all tutorial-related .NET sources.

   Delete the folder containing your *CustdbScripts.cs* and *CustdbScripts.dll* files (*c:\mldnet*).

Note:
Ensure that you do not have other important files in *c:\mldnet*.

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*].

CHAPTER 9

Tutorial: Using .NET and Java for custom authentication

Contents

About this chapter

This chapter guides you through the basic steps for using .NET or Java synchronization scripts for custom authentication.

Introduction

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 10.0
- ♦ Java Software Development Kit

Competencies and experience

You will require:

- ♦ familiarity with Java
- ♦ basic knowledge of MobiLink event scripts

Goals

You will 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 (mlsrv10) with the -sl java 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.

MobiLink server API for .NET

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 .NET server API classes to utilize in your .NET methods. When you compile your .NET class, you reference *iAnywhere.MobiLink.Script.dll*.

MobiLink server API for Java

To execute Java synchronization logic, the MobiLink server must have access to the classes in *mlscript.jar*. *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 your Java or .NET class for custom authentication

1. Create a class called MobiLinkAuth using Java or .NET.

   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.

   For Java, type the following:

```
import ianywhere.ml.script.*;

public class MobiLinkAuth
{

 public void authenticateUser (
  ianywhere.ml.script.InOutInteger authentication_status,
  String user,
  String pwd,
  String newPwd )
 {
   // to do...

 }

}
```

   For .NET, type the following:

```
using iAnywhere.MobiLink.Script;

public class MobiLinkAuth
{
  public void authenticateUser  (
    ref int authentication_status,
    string user,
    string pwd,
    string newPwd)
  {
   // to do...

  }
}
```

2. Write the authenticateUser method.

   This tutorial illustrates a very simple case of custom user authentication. Authentication succeeds if the user name starts with "ML".

   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 118.

   For Java, type the following:

```
public void authenticateUser (
 ianywhere.ml.script.InOutInteger authentication_status,
 String user,
 String pwd,
 String newPwd ) {

if(user.substring(0,1)=="ML") {
        // success: an auth status code of 1000
        authentication_status.setValue(1000);
    } else {
        // fail: an authentication_status code of 4000
        authentication_status.setValue(4000);
    }
}
```

   For .NET, type the following:

```
public void authenticateUser(
 ref int authentication_status,
 string user,
 string pwd,
 string newPwd ) {

    if(user.Substring(0,2)=="ML") {
        // success: an auth status code of 1000
        authentication_status = 1000;
    } else {
        // fail: and authentication_status code of 4000
        authentication_status = 4000;
    }
}
```

3. Compile the MobiLinkAuth class.

   ♦ For .NET, save the file as *MobiLinkAuth.cs* in *c:\mlauth*.

   ♦ At a command prompt, navigate to *c:\mlauth*. Type the following command on a single line to compile the file. Replace *install-dir* with the location of your SQL Anywhere installation.

```
csc /out:c:\mlauth\MobiLinkAuth.dll /target:library
/reference:install-dir\Assembly\v1\iAnywhere.MobiLink.Script.dll"
MobiLinkAuth.cs
```

The *MobiLinkAuth.dll* assembly is generated.

- For Java, save the file as *MobiLinkAuth.java* in *c:\mlauth*. Navigate to *c:\mlauth* at a command prompt. To compile the file, type the following command on a single line. Replace *install-dir* with the location of your SQL Anywhere installation.

  ```
  javac MobiLinkAuth.java -classpath "install-dir/java/mlscript.jar"
  ```

  The *MobiLinkAuth.class* file 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 called SQL Anywhere 10 CustDB.

♦ To register the authenticateUser method for the authenticate_user event

1. Connect to the sample database using Interactive SQL.

   At a command prompt, type:

   ```
   dbisql -c "dsn=SQL Anywhere 10 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, execute the following command in Interactive SQL:

   ```
   call ml_add_java_connection_script(
   'custdb 10.0',
   'authenticate_user',
   'MobiLinkAuth.authenticateUser');
   commit;
   ```

   For .NET, execute the following command in Interactive SQL:

   ```
   call ml_add_dnet_connection_script(
   'custdb 10.0',
   '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" [*MobiLink - Server Administration*]

☞ For more information about the ml_add_dnet_connection_script, see "ml_add_dnet_connection_script" [*MobiLink - Server Administration*].

Lesson 3: Start the MobiLink server for Java or .NET

Starting the MobiLink server with the -sl java or -sl dnet option allows you to specify a set of directories to search for compiled files.

♦ To start the MobiLink server (mlsrv10)

- Connect to the MobiLink sample database and load your Java class or .NET assembly on the mlsrv10 command line.

  For Java, type the following at a command prompt:

  ```
  mlsrv10 -c "dsn=SQL Anywhere 10 CustDB" -sl java(-cp c:\mlauth)
  ```

  For .NET, type the following at a command prompt:

  ```
  mlsrv10 -c "dsn=SQL Anywhere 10 CustDB" -sl dnet(-MLAutoLoadPath=c:
  \mlauth)
  ```

  The MobiLink server window appears. Now, your Java or .NET 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 option" [*MobiLink - Server Administration*].

☞ For more information about starting the MobiLink server for .NET, see "-sl dnet 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. It is a simple sales-status application that you can run 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 10 ► 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 "ML", your Java or .NET logic causes synchronization to fail. Input a value of **50** for the employee ID and press Enter.

   The UltraLite CustDB Demo dialog appears indicating a synchronization error with SQL Code -103. SQL Code -103 represents an invalid user ID or password.

Further reading

☞ For more information about the CustDB Windows Application, see "Exploring the CustDB Sample for MobiLink" on page 41.

Cleanup

Remove tutorial materials from your computer.

♦ To remove tutorial materials from your computer

1. Remove the authenticate_user script from the consolidated database.

   - Connect to the MobiLink sample database in Interactive SQL.

     At a command prompt, type:

     ```
     dbisql -c "dsn=SQL Anywhere 10 CustDB"
     ```

   - Remove the authenticate_user script.

     For Java, execute the following command to remove the authenticate_user script:

     ```
     call ml_add_java_connection_script(
     'custdb 10.0',
     'authenticate_user',
     null);
     commit;
     ```

     For .NET, execute the following command to remove the authenticate_user script:

     ```
     call ml_add_dnet_connection_script(
     'custdb 10.0',
     'authenticate_user',
     null);
     commit;
     ```

2. 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.

3. Close Interactive SQL and UltraLite Windows client application.

   Choose Exit from the File menu of each application.

4. Close the SQL Anywhere, MobiLink, and synchronization client windows.

   Right-click each task-bar item and choose Close.

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*].

CHAPTER 10

Tutorial: Using Direct Row Handling

Contents

About this chapter

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.

Introduction

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 10.0.0
- ♦ Java Software Development Kit or the Microsoft .NET Framework

Competencies and experience

You will require:

- ♦ Familiarity with Java or .NET.
- ♦ Basic knowledge of MobiLink event scripts and MobiLink synchronization.

Goals

You will gain competence and familiarity with:

- ♦ The MobiLink server APIs for Java and .NET.
- ♦ Creating methods for MobiLink direct row handling.

Suggested background reading

☞ For more information about MobiLink synchronization, see "Synchronization Basics" on page 21.

☞ For more information about synchronization techniques, see "Synchronization Techniques" [*MobiLink - Server Administration*].

☞ For more information about direct row handling, see "Direct Row Handling" [*MobiLink - Server Administration*].

Lesson 1: Set up your MobiLink consolidated database

You can use a SQL Anywhere, Adaptive Server Enterprise, Oracle, Microsoft SQL Server, or IBM DB2 MobiLink consolidated database as your 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 will:

- 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 a Database wizard.

♦ To create your SQL Anywhere RDBMS

1. Start Sybase Central.

   Choose Start ▶ Programs ▶ SQL Anywhere 10 ▶ Sybase Central.

   Sybase Central appears.

2. In Sybase Central, choose Tools ▶ SQL Anywhere 10 ▶ Create Database.

   The Create Database wizard dialog appears.

3. Click Next.

4. Leave the default of Create Database on this Computer, and then click Next.

5. Enter the file name and path for the database. For example, enter:

   c:\MLdirect\MLconsolidated.db

6. Follow the remaining instructions in the wizard, accepting the default values, except as follows.

   - Uncheck the option to Install jConnect Meta-Information Support.
   - Uncheck the option to Stop the Database After Last Disconnect.

7. Click Finish.

The database called MLconsolidated appears in Sybase Central.

Define an ODBC data source for the consolidated database.

Use the SQL Anywhere 10 driver to define an ODBC data source for the MLconsolidated database.

♦ To define an ODBC data source for the consolidated database

1. Start the ODBC Administrator:

   From the Sybase Central Tools menu, choose SQL Anywhere 10 ► Open ODBC Administrator.

   The ODBC Data Source Administrator appears.

2. On the User DSN tab, click Add.

   The Create New Data Source dialog appears.

3. Select SQL Anywhere 10, and then click Finish.

   The ODBC Configuration for SQL Anywhere 10 dialog appears.

4. On the ODBC tab, type the Data source name **mobilink_db**. On the Login tab, type **DBA** for the User ID and **SQL** for the Password. On the Database tab, type **MLconsolidated** for the Server Name and *c:\MLdirect\MLconsolidated.db* for the Database file.

5. Click OK to define the data source and OK again to close ODBC Administrator.

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, click your database **MLconsolidated - DBA**. From the Sybase Central File menu, choose Open Interactive SQL.

- ♦ To start Interactive SQL at a command prompt, type the following command:

  ```
  dbisql -c "dsn=mobilink_db"
  ```

2. Run the following command 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 following procedure.

Run MobiLink setup scripts

You can find setup scripts for each supported consolidated database (SQL Anywhere, Adaptive Server Enterprise, Oracle, Microsoft SQL Server, and IBM DB2) in the *MobiLink/setup* subdirectory of your SQL Anywhere 10 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.

   At a command prompt, type the following command:

   ```
   dbisql -c "dsn=mobilink_db"
   ```

2. Run the following command in Interactive SQL to create MobiLink system tables and stored procedures. Replace *c:\Program Files\SQL Anywhere 10* with the location of your SQL Anywhere 10 installation.

   ```
   read "c:\Program Files\SQL Anywhere 10\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 "The Initialization utility" [*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 2: Add synchronization scripts

In this lesson you add scripts to your consolidated database for SQL-based row handling and direct row handling.

SQL-based synchronization

SQL-based synchronization 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** To define how new orders inserted in a remote client database should be applied to the consolidated database.
- ♦ **download_cursor** To define what orders updated in the MobiLink consolidated database should be downloaded to remote clients.

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.

   At a command prompt, type the following command:

   ```
   dbisql -c "dsn=mobilink_db"
   ```

2. Use the ml_add_table_script stored procedure to add SQL-based table scripts for the upload_insert and download_cursor events.

   Run the following commands 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 >= ?');

   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, and end_download events. In "Lesson 3: Write Java or .NET logic for processing direct row handling" on page 132 you create your Java or .NET class.

♦ To add information for direct row handling in MobiLink system tables

1. Connect to your consolidated database in Interactive SQL:

   At a command prompt, type the following command:

   ```
   dbisql -c "dsn=mobilink_db"
   ```

2. Register a Java or .NET method for the end_download event.

   You use this method to free i/o resources when the end_download connection event fires.

   For Java, execute the following command in Interactive SQL.

   ```
   CALL ml_add_java_connection_script( 'default',
      'end_download',
        'MobiLinkOrders.EndDownload' );
   ```

   For .NET, execute the following command 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 synchronization events.

   For Java, execute the following commands 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' );

   commit
   ```

   For .NET, execute the following commands 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' );

   commit
   ```

Interactive SQL registers the user-defined GetUpload and SetDownload methods for the handle_UploadData and handle_DownloadData events, respectively.

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 "Setting 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: Write Java or .NET logic for processing 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. Set download uses the *orderResponses.txt* file to download responses to remote clients.

You also add the EndDownload method to handle the end_download event.

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 listing (Java)" on page 138 or "Complete MobiLinkOrders listing (.NET)" on page 140.

♦ To create a Java or .NET class for direct row handling

1. Create a class called MobiLinkOrders using Java or .NET.

   For Java, type the following code in a text editor or development environment.

```
import ianywhere.ml.script.*;
import java.io.*;
import java.sql.*;

public class  MobiLinkOrders{

// to do...
```

   For .NET, use the following code:

```
using iAnywhere.MobiLink.Script;
using System.IO;
using System.Data;
using System.Text;

public class MobiLinkOrders

// to do...
```

2. Declare a class-level DBConnectionContext instance.

   In Java:

```
// class level DBConnectionContext
    DBConnectionContext _cc;
```

   In .NET:

```
// 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. Create your class constructor.

   Your class constructor sets your class-level DBConnectionContext instance.

   For Java:

```
public MobiLinkOrders( DBConnectionContext cc )
{
 // set your class-level DBConnectionContext
 _cc = cc;
}
```

   For .NET:

```
public MobiLinkOrders( DBConnectionContext cc )
{
    _cc = cc;
}
```

4. Declare objects used for file input and output.

   For Java, declare a java.io.FileWriter and java.io.BufferedReader:

```
// java objects for file i/o
    FileWriter my_writer;
    BufferedReader my_reader;
```

   For .NET, declare a Stream Writer and Stream Reader:

```
// instances for file I/O
private static StreamWriter   my_writer = null;
private static StreamReader   my_reader = null;
```

5. Write the EndDownload method.

   This method handles the end_download connection event and gives you an opportunity to free resources.

   For Java:

```
public void EndDownload() throws IOException
{
  // free i/o resources
    if (my_reader!=null) my_reader.close();
    if (my_writer!=null) my_writer.close();
}
```

   For .NET:

```
public void EndDownload()
{
  if( my_writer != null ) {
        my_writer.Close();
        my_writer = null;
  }
```

6. 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 5: Set up your MobiLink client" on page 144. 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:

```
//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();

   }

// writes out comment details to file
public void writeOrderComment( int _commentID, int _orderID, String
_comments )
   throws IOException
    {

      // a FileWriter for writing order comments
      if(my_writer == null)
      {
        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();

    }
```

   In .NET:

```
// 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 upload 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 turn on on the remote.
       // in this example you just use the known column order to determine
the column
        // indexes

        // only process this insert of 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();
    }
```

7. 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:

      ```
      DownloadData download_d = _cc.getDownloadData();
      DownloadTableData download_td = download_d.getDownloadTableByName
      ( "OrderComments" );
      ```

      For .NET:

      ```
      DownloadTableData comments_for_download =
          _cc.GetDownloadData().GetDownloadTableByName( "OrderComments" );
      ```

      Note:
      You create this table on the remote database in "Lesson 5: Set up your MobiLink client" on page 144.

   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.

```
PreparedStatement update_ps = download_td.getUpsertPreparedStatement
();
```

For .NET, use the DownloadTableData GetUpsertCommand method:

```
// you will 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. Send down responses to remote clients.

Create a text file called *orderResponses.txt* in *c:\MLdirect*. 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, we will ship promotional material.
787  35  Yes, the product is going out of production.
788  36  No, we can't increase your commission...
...
```

d. Set the download data for each row.

For Java, the following example traverses through the *orderResponses.txt* and adds data to the MobiLink download.

For Java:

```
//   a BufferedReader for writing out responses
if (my_reader==null)
 my_reader = new BufferedReader(new FileReader( "c:\\MLdirect\
\orderResponses.txt"));

// send updated comments down to clients
String commentLine;

// read the first line from orderResponses.txt
commentLine = my_reader.readLine();

   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[0]);
   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 from orderResponses.txt
         commentLine  = my_reader.readLine();
      }
```

For .NET:

```
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 IDataParameters
        ((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();
}
```

e. Close the prepared statement used for adding insert or update operations to the download.

For Java:

```
update_ps.close();
```

For .NET, you do not need to close the IDBCommand. It is destroyed by MobiLink at the end of the download.

8. Compile your class file.

♦ Navigate to the directory containing your Java or .NET source files.

♦ Compile MobiLinkOrders with references to the MobiLink server API library for Java or .NET.

For Java, you need to reference *mlscript.jar* in the *java* subdirectory of your SQL Anywhere installation. Type the following command line to compile your Java class, replacing *c:\Program Files\SQL Anywhere 10* with your SQL Anywhere 10 directory.

```
javac -classpath "c:\Program Files\SQL Anywhere 10\java\mlscript.jar"
MobiLinkOrders.java
```

For .NET, use the following command:

```
csc /out:MobiLinkServerCode.dll /target:library /reference:"c:\Program
Files\SQL Anywhere 10\Assembly\v1\iAnywhere.MobiLink.Script.dll"
MobiLinkOrders.cs
```

Further reading

☞ For more information about class constructors and DBConnectionContext, see "Constructors" [*MobiLink - Server Administration*] for .NET synchronization logic or "Constructors" [*MobiLink - Server Administration*] for Java synchronization logic.

☞ 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 listing (Java)

Following is the complete MobiLinkOrders listing for Java direct row handling. For a step by step explanation, see "Lesson 3: Write Java or .NET logic for processing direct row handling" on page 132.

```
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 writing out 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 listing (.NET)

Following is the complete MobiLinkOrders listing for .NET object-based uploads and downloads. For a step by step explanation, see "Lesson 3: Write Java or .NET logic for processing direct row handling" on page 132.

```
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:\\mlobjbased\\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 upload 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 turn on on the remote.
        // in this example you just use the known column order to determine
the column
        // indexes

        // only process this insert of 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:\\mlobjbased\
\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( "Nothing to do in SetDownload() there is no
file to read." );
        return;
    }
    DownloadTableData comments_for_download =
        _cc.GetDownloadData().GetDownloadTableByName( "OrderComments" );

    // you will 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 IDataParameters
                ((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 4: Start the MobiLink server

In this lesson you start the MobiLink server. You start the MobiLink server (mlsrv10) using the -c option to connect to your consolidated database and the -sl java or -sl dnet option to load your Java or .NET class, respectively.

♦ To start the MobiLink server for direct row handling

- Connect to your consolidated database and load .NET or Java classes on the mlsrv10 command line.

  For Java, type the following command. Replace *c:\MLdirect* with the location of your Java source files.

  ```
  mlsrv10 -c "dsn=mobilink_db" -o serverOut.txt -v+ -dl -zu+ -x tcpip  -sl
  java (-cp c:\MLdirect)
  ```

  For .NET:

  ```
  mlsrv10 -c "dsn=mobilink_db;uid=DBA;pwd=sql" -o serverOut.txt -v+ -dl -zu
  + -x tcpip  -sl dnet (-Dautoloadpath=c:\MLdirect)
  ```

  The MobiLink server 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 Virtual Machine 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 option" [*MobiLink - Server Administration*] and "-sl dnet option" [*MobiLink - Server Administration*], respectively.

Lesson 5: Set up your MobiLink client

In this tutorial, 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.

   In this lesson, you create a SQL Anywhere database using the dbinit command line utility.

   ♦ To create your SQL Anywhere database, at a command prompt, type the following command:

     ```
     dbinit -i -k remote1
     ```

     The -i and -k options tell dbinit to omit Sybase jConnect support and Watcom SQL compatibility views, respectively.

   ♦ To start the database engine, type:

     ```
     dbeng10 remote1
     ```

2. Connect to your MobiLink client database using Interactive SQL.

   From a command prompt, type:

   ```
   dbisql -c "eng=remote1;uid=DBA;pwd=sql"
   ```

3. Create the RemoteOrders table.

   Execute the following command 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)
   )
   ```

4. Run the following command in Interactive SQL to create the OrderComments table:

   ```
   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)
    )
```

5. Create your MobiLink synchronization user, publication, and subscription:

   ```
   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.

Further reading

☞ For information about creating a SQL Anywhere database, see “The Initialization utility” [*SQL Anywhere Server - Database Administration*].

☞ For information about MobiLink clients, see “Introducing 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 6: 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:

   At a command prompt, type

   ```
   dbisql -c "eng=remote1;uid=DBA;pwd=sql"
   ```

2. Add an order to the RemoteOrders table in the client database.

   Execute the following command 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.

   Execute the following command 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.

   Execute the following in Interactive SQL:

   ```
   COMMMIT;
   ```

♦ To start the synchronization client (client-side)

- At the command prompt, type the following on a single line:

  ```
  dbmlsync -c "eng=remote1;uid=DBA;pwd=sql" -e scn=on -o rem1.txt -v+
  ```

  Below is a description of each option:

Option	Description
-c	Precedes 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. 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.

   ```
   1   1   Promotional material shipped
   ```

2. Run synchronization using the dbmlsync client utility.

   This action takes place on the client-side.

   At a command prompt, type the following on a single line.

   ```
   dbmlsync -c "eng=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 mlsrv10 -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:
   - Start the ODBC Administrator.

     Type the following at a command prompt:

     ```
     odbcad32
     ```
   - Delete the mobilink_db data source.
4. Delete the consolidated and remote databases:
   - Navigate to the directory containing your consolidated and remote databases.
   - 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 direct row handling, see “Direct Row Handling” [*MobiLink - Server Administration*].

Index