



MobiLink Synchronization User's Guide

Part number: 38132-01-0900-01

Last modified: June 2003

Copyright © 1989–2003 Sybase, Inc. Portions copyright © 2001–2003 iAnywhere Solutions, Inc. All rights reserved.

No part of this publication may be reproduced, transmitted, or translated in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without the prior written permission of iAnywhere Solutions, Inc. iAnywhere Solutions, Inc. is a subsidiary of Sybase, Inc.

Sybase, SYBASE (logo), AccelaTrade, ADA Workbench, Adaptable Windowing Environment, Adaptive Component Architecture, Adaptive Server, Adaptive Server Anywhere, Adaptive Server Enterprise, Adaptive Server Enterprise Monitor, Adaptive Server Enterprise Replication, Adaptive Server Everywhere, Adaptive Server IQ, Adaptive Warehouse, AnswerBase, Anywhere Studio, Application Manager, AppModeler, APT Workbench, APT-Build, APT-Edit, APT-Execute, APT-Library, APT-Translator, ASEP, AvantGo, AvantGo Application Alerts, AvantGo Mobile Delivery, AvantGo Mobile Document Viewer, AvantGo Mobile Inspection, AvantGo Mobile Marketing Channel, AvantGo Mobile Pharma, AvantGo Mobile Sales, AvantGo Pylon, AvantGo Pylon Application Server, AvantGo Pylon Conduit, AvantGo Pylon PIM Server, AvantGo Pylon Pro, Backup Server, BayCam, Bit-Wise, BizTracker, Certified PowerBuilder Developer, Certified SYBASE Professional, Certified SYBASE Professional (logo), ClearConnect, Client Services, Client-Library, CodeBank, Column Design, ComponentPack, Connection Manager, Convoy/DM, Copernicus, CSP, Data Pipeline, Data Workbench, DataArchitect, Database Analyzer, DataExpress, DataServer, DataWindow, DB-Library, dbQueue, Developers Workbench, Direct Connect Anywhere, DirectConnect, Distribution Director, Dynamic Mobility Model, Dynamo, e-ADK, E-Anywhere, e-Biz Integrator, E-Whatever, EC Gateway, ECMAP, ECRTP, eFulfillment Accelerator, Electronic Case Management, Embedded SQL, EMS, Enterprise Application Studio, Enterprise Client/Server, Enterprise Connect, Enterprise Data Studio, Enterprise Manager, Enterprise Portal (logo), Enterprise SQL Server Manager, Enterprise Work Architecture, Enterprise Work Designer, Enterprise Work Modeler, eProcurement Accelerator, eremote, Everything Works Better When Everything Works Together, EWA, Financial Fusion, Financial Fusion (and design), Financial Fusion Server, Formula One, Fusion Powered e-Finance, Fusion Powered Financial Destinations, Fusion Powered STP, Gateway Manager, GeoPoint, GlobalFIX, iAnywhere, iAnywhere Solutions, ImpactNow, Industry Warehouse Studio, InfoMaker, Information Anywhere, Information Everywhere, InformationConnect, InstaHelp, InternetBuilder, iremote, iScript, Jaguar CTS, jConnect for JDBC, KnowledgeBase, Logical Memory Manager, M-Business Channel, M-Business Network, M-Business Server, Mail Anywhere Studio, MainframeConnect, Maintenance Express, Manage Anywhere Studio, MAP, MDI Access Server, MDI Database Gateway, media.splash, Message Anywhere Server, MetaWorks, MethodSet, ML Query, MobicATS, My AvantGo, My AvantGo Media Channel, My AvantGo Mobile Marketing, MySupport, Net-Gateway, Net-Library, New Era of Networks, Next Generation Learning, Next Generation Learning Studio, O DEVICE, OASIS, OASIS (logo), ObjectConnect, ObjectCycle, OmniConnect, OmniSQL Access Module, OmniSQL Toolkit, Open Biz, Open Business Interchange, Open Client, Open Client/Server, Open Client/Server Interfaces, Open ClientConnect, Open Gateway, Open Server, Open ServerConnect, Open Solutions, Optima++, Partnerships that Work, PB-Gen, PC APT Execute, PC DB-Net, PC Net Library, PhysicalArchitect, Pocket PowerBuilder, PocketBuilder, Power Through Knowledge, Power++, power.stop, PowerAMC, PowerBuilder, PowerBuilder Foundation Class Library, PowerDesigner, PowerDimensions, PowerDynamo, Powering the New Economy, PowerJ, PowerScript, PowerSite, PowerSocket, Powersoft, Powersoft Portfolio, Powersoft Professional, PowerStage, PowerStudio, PowerTips, PowerWare Desktop, PowerWare Enterprise, ProcessAnalyst, QAnywhere, Rapport, Relational Beans, RepConnector, Replication Agent, Replication Driver, Replication Server, Replication Server Manager, Replication Toolkit, Report Workbench, Report-Execute, Resource Manager, RW-DisplayLib, RW-Library, S.W.I.F.T. Message Format Libraries, SAFE, SAFE/PRO, SDF, Secure SQL Server, Secure SQL Toolset, Security Guardian, SKILS, smart.partners, smart.parts, smart.script, SQL Advantage, SQL Anywhere, SQL Anywhere Studio, SQL Code Checker, SQL Debug, SQL Edit, SQL Edit/TPU, SQL Everywhere, SQL Modeler, SQL Remote, SQL Server, SQL Server Manager, SQL Server SNMP SubAgent, SQL Server/CFT, SQL Server/DBM, SQL SMART, SQL Station, SQL Toolset, SQLJ, Stage III Engineering, Startup.Com, STEP, SupportNow, Sybase Central, Sybase Client/Server Interfaces, Sybase Development Framework, Sybase Financial Server, Sybase Gateways, Sybase Learning Connection, Sybase MPP, Sybase SQL Desktop, Sybase SQL Lifecycle, Sybase SQL Workgroup, Sybase Synergy Program, Sybase User Workbench, Sybase Virtual Server Architecture, SybaseWare, Syber Financial, SyberAssist, SybMD, SyBooks, System 10, System 11, System XI (logo), SystemTools, Tabular Data Stream, The Enterprise Client/Server Company, The Extensible Software Platform, The Future Is Wide Open, The Learning Connection, The Model For Client/Server Solutions, The Online Information Center, The Power of One, TradeForce, Transact-SQL, Translation Toolkit, Turning Imagination Into Reality, UltraLite, UltraLite.NET, UNIBOM, Unilib, Uninull, Unisep, Unistring, URK Runtime Kit for UniCode, Versacore, Viewer, VisualWriter, VQL, Warehouse Control Center, Warehouse Studio, Warehouse WORKS, WarehouseArchitect, Watcom, Watcom SQL, Watcom SQL Server, Web Deployment Kit, Web.PB, Web.SQL, WebSights, WebViewer, WorkGroup SQL Server, XA-Library, XA-Server, and XP Server are trademarks of Sybase, Inc. or its subsidiaries.

Certicom, MobileTrust, and SSL Plus are trademarks and Security Builder is a registered trademark of Certicom Corp. Copyright © 1997–2001 Certicom Corp. Portions are Copyright © 1997–1998, Consensus Development Corporation, a wholly owned subsidiary of Certicom Corp. All rights reserved. Contains an implementation of NR signatures, licensed under U.S. patent 5,600,725. Protected by U.S. patents 5,787,028; 4,745,568; 5,761,305. Patents pending.

All other trademarks are property of their respective owners.

Contents

About This Manual	ix
SQL Anywhere Studio documentation	x
Documentation conventions	xiii
The CustDB sample database	xv
Finding out more and providing feedback	xvi
 I Using MobiLink Technology	 1
1 Introducing MobiLink Synchronization	3
The MobiLink synchronization process	4
2 Synchronization Basics	7
Parts of the synchronization system	8
Consolidated database	10
The MobiLink synchronization server	16
MobiLink clients	19
The synchronization process	21
Upload-only and download-only synchronization	30
Options for writing synchronization logic	31
Security	35
3 Writing Synchronization Scripts	37
Introduction to synchronization scripts	38
Scripts and the synchronization process	44
Script types	46
Script parameters	48
Script versions	49
Adding and deleting scripts in your consolidated database	51
Writing scripts to upload rows	54
Writing scripts to download rows	56
Writing scripts to handle errors	62
Testing script syntax	64
DBMS-dependent scripts	65
4 Synchronization Techniques	69
Introduction	70
Development tips	71
Timestamp-based synchronization	72


Snapshot synchronization	74
Partitioning rows among remote databases	77
Maintaining unique primary keys	81
Handling conflicts	90
Data entry	94
Handling deletes	95
Handling failed downloads	96
Downloading a result set from a stored procedure call	97
Schema changes in remote databases	100
5 Authenticating MobiLink Users	103
About MobiLink users	104
Choosing a user authentication mechanism	107
User authentication architecture	108
Providing initial passwords for users	110
Synchronizations from new users	111
Prompting end users to enter passwords	112
Changing passwords	113
Custom user authentication	114
6 File-Based Downloads	117
Introduction	118
Setting up file-based downloads	119
Validation checks	123
Examples	127
7 Server-Initiated Synchronization	137
Introduction	138
Supported platforms	141
Setting up server-initiated synchronization	142
Push requests	143
Set up the Notifier	145
Set up the Listener	154
Listener Software Development Kit	162
Deployment considerations	163
Walkthrough of server-initiated synchronization	164
Sample applications	166
8 Adaptive Server Anywhere Clients	167
Creating a remote database	168
Publishing data	171
Creating MobiLink users	178
Subscribing MobiLink synchronization users	182
Initiating synchronization	185

Using ActiveSync synchronization	189
Temporarily stopping synchronization of deletes	193
Customizing the client synchronization process	194
Scheduling synchronization	198
Adaptive Server Anywhere version 7 MobiLink clients	200
9 UltraLite Clients	207
Introduction to synchronization streams	208
Synchronizing UltraLite databases on the Palm Computing Platform	209
Synchronizing UltraLite databases on Windows CE	223
10 Writing Synchronization Scripts in Java	227
Introduction	228
Setting up Java synchronization logic	229
Running Java synchronization logic	231
Writing Java synchronization logic	232
Java synchronization example	240
MobiLink Java API Reference	246
11 Writing Synchronization Scripts in .NET	251
Introduction	252
Setting up .NET synchronization logic	253
Running .NET synchronization logic	255
Writing .NET synchronization logic	260
.NET synchronization example	266
MobiLink .NET API Reference	269
12 MobiLink Performance	285
Performance tips	286
Key factors influencing MobiLink performance	290
Monitoring MobiLink performance	295
13 MobiLink Monitor	297
Introduction	298
Starting the MobiLink Monitor	299
Using the MobiLink Monitor	302
Saving Monitor data	307
Customizing your statistics	308
MobiLink statistical properties	310
14 Synchronizing Through a Web Server	313
Introduction	314
Setting up the Redirector	315
Configuring MobiLink clients and servers for the Redirector	316

Configuring Redirector properties (all versions)	318
Configuring an NSAPI Redirector for Netscape web servers	320
Configuring an ISAPI Redirector for Microsoft web servers	323
Configuring the servlet Redirector	325
15 Running MobiLink Outside the Current Session	329
Running the UNIX MobiLink server as a daemon	330
Running the Windows MobiLink server as a service	331
Troubleshooting MobiLink server startup	336
16 Transport-Layer Security	337
About transport-layer security	338
Invoking transport-layer security	346
Certificate authorities	351
Certificate chains	352
Enterprise root certificates	353
Globally signed certificates	358
Obtaining server-authentication certificates	360
Verifying certificate fields	363
II MobiLink Tutorials	367
17 Tutorial: Synchronizing Adaptive Server Anywhere Databases	369
Introduction	370
Lesson 1: Creating and populating your databases	371
Lesson 2: Running the MobiLink synchronization server	375
Lesson 3: Running the MobiLink synchronization client	377
Tutorial cleanup	379
Summary	380
Further reading	381
18 Tutorial: Writing SQL Scripts Using Sybase Central	383
Introduction	384
Lesson 1: Creating your databases	385
Lesson 2: Creating scripts for your synchronization	389
Lesson 3: Running the MobiLink synchronization server	392
Lesson 4: Running the MobiLink synchronization client	394
Lesson 5: Monitoring your MobiLink synchronization using log files	396
Tutorial cleanup	398
Further reading	399
19 Tutorial: Using MobiLink with an Oracle 8i Consolidated Database	401
Introduction	402

Lesson 1: Create your databases	403
Lesson 2: Starting the MobiLink synchronization server	409
Lesson 3: Running the MobiLink synchronization client	410
Summary	411
Further reading	412
20 The Contact Sample Application	413
Introduction	414
Setup	415
Tables in the Contact databases	417
Users in the Contact sample	420
Synchronization	421
Monitoring statistics and errors in the Contact sample	428
21 The CustDB Sample Application	429
Introduction	430
Setup	432
Tables in the CustDB databases	440
Users in the CustDB sample	443
Synchronization	444
Maintaining the customer and order primary key pools	448
Further reading	450
Index	451

About This Manual

Subject	This manual describes MobiLink, a session-based relational-database synchronization system. MobiLink technology allows two-way replication and is well suited to mobile computing environments.
Audience	This manual is for users of Adaptive Server Anywhere and other relational database systems who wish to add synchronization or replication to their information systems.
Before you begin	 For a comparison of MobiLink with other synchronization and replication technologies, see “Replication Technologies” [<i>Introducing SQL Anywhere Studio</i> , page 19].

SQL Anywhere Studio documentation

The SQL Anywhere Studio documentation

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

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

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

- ◆ **Adaptive Server Anywhere Error Messages** This book provides a complete listing of Adaptive Server Anywhere error messages together with diagnostic information.
- ◆ **SQL Anywhere Studio Security Guide** This book provides information about security features in Adaptive Server Anywhere databases. Adaptive Server Anywhere 7.0 was awarded a TCSEC (Trusted Computer System Evaluation Criteria) C2 security rating from the U.S. Government. This book may be of interest to those who wish to run the current version of Adaptive Server Anywhere in a manner equivalent to the C2-certified environment.
- ◆ **MobiLink Synchronization User's Guide** This book describes how to use the MobiLink data synchronization system for mobile computing, which enables sharing of data between a single Oracle, Sybase, Microsoft or IBM database and many Adaptive Server Anywhere or UltraLite databases.
- ◆ **MobiLink Synchronization Reference** This book is a reference guide to MobiLink command line options, synchronization scripts, SQL statements, stored procedures, utilities, system tables, and error messages.
- ◆ **iAnywhere Solutions ODBC Drivers** This book describes how to set up ODBC drivers to access consolidated databases other than Adaptive Server Anywhere from the MobiLink synchronization server and from Adaptive Server Anywhere remote data access.
- ◆ **SQL Remote User's Guide** This book describes all aspects of the SQL Remote data replication system for mobile computing, which enables sharing of data between a single Adaptive Server Anywhere or Adaptive Server Enterprise database and many Adaptive Server Anywhere databases using an indirect link such as e-mail or file transfer.
- ◆ **SQL Anywhere Studio Help** This book includes the context-sensitive help for Sybase Central, Interactive SQL, and other graphical tools. It is not included in the printed documentation set.
- ◆ **UltraLite Database User's Guide** This book is intended for all UltraLite developers. It introduces the UltraLite database system and provides information common to all UltraLite programming interfaces.
- ◆ **UltraLite Interface Guides** A separate book is provided for each UltraLite programming interface. Some of these interfaces are provided as UltraLite components for rapid application development, and others are provided as static interfaces for C, C++, and Java development.

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

Documentation formats SQL Anywhere Studio provides documentation in the following formats:

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

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

To access the online documentation on UNIX operating systems, see the HTML documentation under your SQL Anywhere installation.

- ◆ **Printable books** The SQL Anywhere books are provided as a set of PDF files, viewable with Adobe Acrobat Reader.

The PDF files are available on the CD ROM in the *pdf_docs* directory. You can choose to install them when running the setup program.

- ◆ **Printed books** The complete set of books is available from Sybase sales or from eShop, the Sybase online store. You can access eShop by clicking How to Buy ► eShop at <http://www.ianywhere.com>.

Documentation conventions

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

Syntax conventions

The following conventions are used in the SQL syntax descriptions:

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

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

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

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

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

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

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

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

RELEASE SAVEPOINT [*savepoint-name*]

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

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

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

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

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

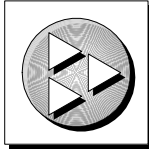
[**QUOTES** { **ON** | **OFF** }]

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

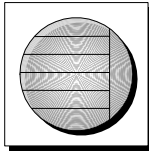
Graphic icons

The following icons are used in this documentation.

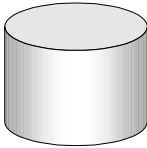
- ◆ A client application.



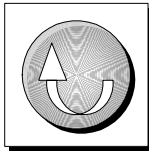
- ◆ A database server, such as Sybase Adaptive Server Anywhere.



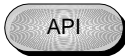
- ◆ A database. In some high-level diagrams, the icon may be used to represent both the database and the database server that manages it.



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



- ◆ A programming interface.



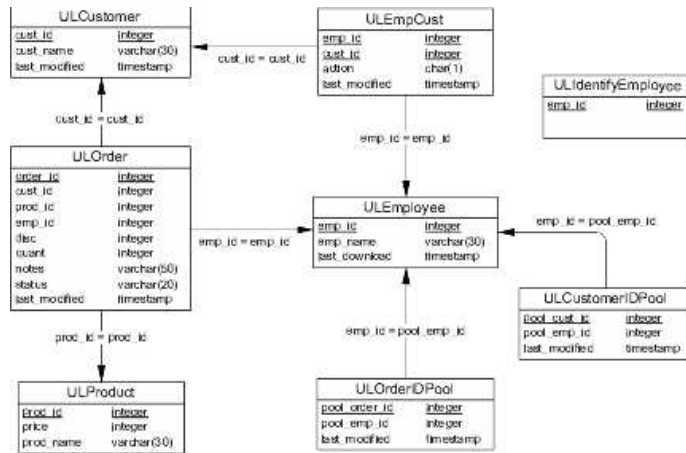
The CustDB sample database

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

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

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

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



Finding out more and providing feedback

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

You can provide feedback on this documentation and on the software through newsgroups set up to discuss SQL Anywhere technologies. These newsgroups can be found 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.

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 insure 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.

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

About this chapter

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

☞ For hands-on tutorials introducing MobiLink, see

- ◆ [“Tutorial: Synchronizing Adaptive Server Anywhere Databases” on page 369](#)
- ◆ [“Tutorial: Writing SQL Scripts Using Sybase Central” on page 383](#)
- ◆ [“Tutorial: Using MobiLink with an Oracle 8i Consolidated Database” on page 401](#)
- ◆ [“The Contact Sample Application” on page 413](#)
- ◆ [“The CustDB Sample Application” on page 429](#)

☞ For a more detailed introduction to MobiLink technology, see [“Synchronization Basics” on page 7](#).

Contents

Topic:	page
The MobiLink synchronization process	4

The MobiLink synchronization process

MobiLink is a session-based synchronization system that allows two-way synchronization between a main database, called the consolidated database, and many remote databases. The consolidated database, which can be one of several ODBC-compliant databases, holds the master copy of all the data. Remote databases can be either Adaptive Server Anywhere or UltraLite databases.

Synchronization typically begins when a MobiLink remote site opens a connection to a MobiLink synchronization server. During synchronization, the MobiLink client at the remote site uploads database changes that were made to the remote database since the previous synchronization. On receiving this data, the MobiLink synchronization server updates the consolidated database, and then downloads changes on the consolidated database to the remote database.

MobiLink features

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

- ◆ **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, specify what gets downloaded, and handle different schema and names between the consolidated and remote databases.
- ◆ **Automation** MobiLink has a number of automated capabilities. The MobiLink synchronization server can be instructed to generate scripts suitable for snapshot synchronization, or instructed to generate example synchronization scripts. It can also automatically add users for authentication. Server-initiated synchronization allows you to push data updates to remote databases.
- ◆ **Monitoring and reporting** MobiLink provides two mechanisms for monitoring your synchronizations: the MobiLink Monitor, and statistical scripts. You can monitor scripts, schema contents, row-count values, script names, translated script contents, and row values.
- ◆ **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, number of worker threads, logging verbosity, or BLOB cache size.

- ◆ **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.
- ◆ **File-based download** Downloads can be distributed as files, enabling offline distribution of synchronization changes. This allows you to create a file once and distribute it widely.
- ◆ **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 communication streams** Synchronization can be carried out over TCP/IP, HTTP, or HTTPS. Palm devices can synchronize through HotSync. Windows CE devices can synchronize using ActiveSync.
- ◆ **Remote-initiated** Synchronization between a remote database and a consolidated database can be initiated at the remote database.
- ◆ **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.
- ◆ **Transactional integrity** 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: Adaptive Server Anywhere, Adaptive Server Enterprise, Oracle, Microsoft SQL Server, or IBM DB2. Remote databases can be Adaptive Server Anywhere or UltraLite databases. The MobiLink synchronization server runs on Windows or UNIX platforms. Adaptive Server Anywhere runs on Windows, Windows CE, or UNIX machines. UltraLite runs on Palm, Windows CE, or Java-based devices.

-
- ◆ **Flexibility** The MobiLink synchronization server uses SQL, Java, or .NET scripts to control the upload and download of data. The scripts are executed according to an event model during each synchronization. Event-based scripting provides great flexibility in the design of the synchronization process, including such features as conflict resolution, error reporting, and user authentication.
 - ◆ **Scalability and performance** MobiLink synchronization is scalable: a single server can handle thousands of simultaneous synchronizations, and multiple MobiLink servers can be run simultaneously using load balancing. The MobiLink synchronization server is multi-threaded and uses connection pooling with the consolidated database. MobiLink provides extensive monitoring and reporting facilities.
 - ◆ **Easy to get started** Simple MobiLink installations can be constructed quickly. More complex refinements can be added incrementally for full-scale production work.

CHAPTER 2

Synchronization Basics

About this chapter

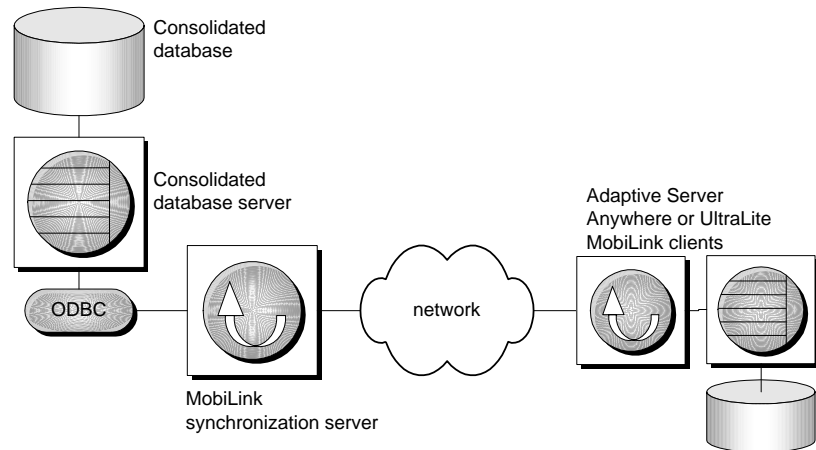
This chapter introduces the basic components of MobiLink technology and provides information about how to set up your synchronization system.

Contents

Topic:	page
Parts of the synchronization system	8
Consolidated database	10
The MobiLink synchronization server	16
MobiLink clients	19
The synchronization process	21
Upload-only and download-only synchronization	30
Options for writing synchronization logic	31
Security	35

Parts of the synchronization system

The following diagram shows the major parts of the synchronization system.



- ◆ **consolidated database** This database contains the central copy of all information in the synchronization system.
☞ For more information, see [“Consolidated database” on page 10](#).
- ◆ **consolidated database server** The server, or DBMS, that manages the consolidated database. This server can be a Sybase product, such as Adaptive Server Anywhere or Adaptive Server Enterprise, or it may be a supported system made by another company.
☞ For more information, see [“Supported consolidated databases” on page 10](#).
- ◆ **ODBC connection** All communication between the MobiLink synchronization server and the consolidated database occurs through an ODBC connection. ODBC allows the synchronization server to utilize a variety of consolidated database systems.
☞ For more information, see [“ODBC Drivers”](#) [*MobiLink Synchronization Reference*, page 335].
- ◆ **MobiLink synchronization server** This server manages the synchronization process and provides the interface between all MobiLink clients and the consolidated database server.
☞ For more information, see [“The MobiLink synchronization server” on page 16](#).

- ◆ **Network** The connection between the MobiLink synchronization server, dbmlsrv9, and the MobiLink client, dbmlsync or UltraLite, can use a number of protocols.

☞ For more information about connecting to dbmlsync, see “-x option” [*MobiLink Synchronization Reference*, page 24]. For information about connecting to UltraLite, see “Stream parameters reference” [*UltraLite Database User’s Guide*, page 179].

- ◆ **MobiLink client** The client can be installed on a handheld device such as a Palm Pilot or PocketPC, a server or desktop computer, or an embedded device such as a cell phone or vending machine. Two types of clients are supported: UltraLite and Adaptive Server Anywhere databases. Either or both may be used in a single MobiLink installation.

☞ For more information, see [“MobiLink clients” on page 19](#).

Consolidated database

Applications synchronize with a central, consolidated database. This database is the master repository of information in the synchronization system.

There are many ways to structure the relations between consolidated and remote databases. Following are two examples.

The schema of the remote databases can be a subset of the schema of the consolidated database. For example, a table called emp might be repeated among a number of different remote sites, and the consolidated database might use column data from emp.salary in a table called expense. In this instance, the schemas of the consolidated and remote databases are different, though data is shared.

The schema of the remote database can also be parallel in structure to the schema of the consolidated database. Here, the schema of the consolidated database is a reference for the remote database. In the consolidated database, you may already have tables that correspond to each of the remote tables. In this instance, the schemas in the consolidated and remote databases are virtually the same, and the data in the remote is only a subset of the data on the consolidated.

You write **synchronization scripts** for each table in the remote database and you save these scripts on the consolidated database. These scripts, from their central location on the consolidated database, direct the synchronization server in moving data between remote and consolidated databases. One script for a particular remote table tells the synchronization server where to store data uploaded from that remote table in the consolidated database. Another script tells the synchronization server which data to download to the same remote table.

Supported consolidated databases

Your consolidated database can be one of the following ODBC-compliant databases: Adaptive Server Anywhere, Adaptive Server Enterprise, Oracle, IBM DB2, and Microsoft SQL Server. You can use synchronization scripts to exploit the features of your particular consolidated server.

☞ For information about writing synchronization scripts for specific consolidated databases, see [“DBMS-dependent scripts” on page 65](#).

☞ For information about setting up each type of database as a consolidated database, see [“Setting up a consolidated database” on page 11](#).

How remote tables relate to consolidated tables

Arbitrary relationships permitted	<p>Synchronization designs can specify mappings between tables and rows in the remote database with tables and rows in the consolidated database.</p> <p>Tables in a remote database need not be identical to those in the consolidated database. Synchronized data in one remote application table can be distributed between columns in different tables, and even between tables in different consolidated databases. You specify these relationships using synchronization scripts.</p> <p>Synchronization scripts are associated with the consolidated database. SQL scripts are stored in the consolidated database, and Java and .NET scripts are referenced.</p>
Direct relationships are simple	<p>You can often simplify your design using a table structure in the remote database that is a subset of that in the consolidated database. Using this method, every table in the remote database exists in the consolidated database. Corresponding tables have the same structure and foreign key relationships as those in the consolidated database.</p> <p>Tables in the consolidated database will frequently contain extra columns that are not synchronized. Indeed, extra columns can aid synchronization. For example, a timestamp column can identify new or updated rows in the consolidated database. In other cases, extra columns or tables in the consolidated database may hold information that is not required at remote sites.</p>

Setting up a consolidated database

Setup scripts	<p>To set up a database so that it can be used as a MobiLink consolidated database, you must run a setup script that installs MobiLink system tables and stored procedures. The exception is Adaptive Server Anywhere databases, which are preconfigured with the appropriate system tables and stored procedures. For instructions on how to run the setup scripts, see the sections below for each supported RDBMS.</p> <p>☞ For more information about MobiLink system tables, see “MobiLink System Tables” [<i>MobiLink Synchronization Reference</i>, page 315].</p> <p>☞ For more information about stored procedures, see “Stored Procedures” [<i>MobiLink Synchronization Reference</i>, page 261].</p>
ODBC connection	<p>In addition, the MobiLink synchronization server needs an ODBC connection to your consolidated database. You must configure the appropriate ODBC driver for your server and create an ODBC data source for the database on the computer where your MobiLink synchronization</p>

server is running.

☞ For a summary of supported ODBC drivers, see “ODBC Drivers” [*MobiLink Synchronization Reference*, page 335].

☞ For updated information and complete functional specifications, see http://www.ianywhere.com/developer/technotes/odbc_mobilink.html.

☞ For information about configuring ODBC drivers for MobiLink consolidated databases, see “Introduction to ODBC Drivers” [*iAnywhere Solutions ODBC Drivers*, page 1].

☞ For specific information, see the section below for each supported RDBMS.

Setting up a Sybase Adaptive Server Anywhere consolidated database

Adaptive Server Anywhere databases are automatically configured so that they can be used as a MobiLink consolidated database without running a setup script.

A setup script is provided for Adaptive Server Anywhere databases in case you want to examine source code. For example, it includes source code for the `ml_add_connection_script` stored procedure. This setup script is called `syncasa.sql` and it is located in the `scripts` subdirectory of your SQL Anywhere installation.

Setting up the ODBC driver

You must set up an ODBC DSN for your Adaptive Server Anywhere consolidated database. The ODBC driver for Adaptive Server Anywhere is installed with SQL Anywhere Studio.

☞ For information about the Adaptive Server Anywhere ODBC driver, see “Working with ODBC data sources” [*ASA Database Administration Guide*, page 53].

Setting up a Sybase Adaptive Server Enterprise consolidated database

To set up Adaptive Server Enterprise version 12.5 or later to work as a MobiLink consolidated database, run the `syncase125.sql` setup script, located in the `MobiLink\setup` subdirectory of your SQL Anywhere installation. For versions prior to 12.5, run `syncase.sql` from the same location.

Tips

☞ For tips on using Adaptive Server Enterprise as a MobiLink consolidated database, see “DBMS-dependent scripts” on page 65 and “Supported DBMS scripting strategies” on page 65.

ODBC driver

You must set up an ODBC DSN for your Adaptive Server Enterprise consolidated database. SQL Anywhere Studio includes an iAnywhere

Solutions ODBC driver for Adaptive Server Enterprise. You must configure this driver to work with MobiLink.

☞ For more information, see “iAnywhere Solutions ODBC Driver for Sybase Adaptive Server Enterprise” [*iAnywhere Solutions ODBC Drivers*, page 15].

Setting up an Oracle consolidated database

To set up Oracle to work as a MobiLink consolidated database, run the *syncora.sql* setup script, located in the *MobiLink\setup* subdirectory of your SQL Anywhere installation.

Tips

☞ For tips on using Oracle as a MobiLink consolidated database, see “DBMS-dependent scripts” on page 65 and “Supported DBMS scripting strategies” on page 65.

ODBC driver

You must set up an ODBC DSN for your Oracle consolidated database. SQL Anywhere Studio includes an iAnywhere Solutions ODBC driver for Oracle. You must configure this driver to work with MobiLink.

☞ For more information, see “iAnywhere Solutions ODBC Driver for Oracle Wire Protocol” [*iAnywhere Solutions ODBC Drivers*, page 31].

Setting up an IBM DB2 consolidated database

To set up IBM DB2 UDB version 6 or later to work as a MobiLink consolidated database, run the *syncdb2long.sql* setup script, located in the *MobiLink\setup* subdirectory of your SQL Anywhere installation. For IBM DB2 prior to version 6, run the *syncdb2.sql* setup script from the same location.

The *syncdb2.sql* and *syncdb2long.sql* scripts contain a default connection statement, `connect to DB2Database`. You should make a copy of the script and alter this line to be appropriate for your installation. The syntax of the line must be:

```
connect to DB2Database user userid using password
```

where *DB2Database*, *userid*, and *password* are names you provide.

The *syncdb2.sql* and *syncdb2long.sql* scripts use the tilde character (~) as a command delimiter. You can run the scripts as follows:

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

In addition, there are columns that require a LONG tablespace. If there is no default LONG tablespace, the creation statements for the tables containing these columns must be qualified appropriately, as in the following example.

```
CREATE TABLE ... ( ... )  
IN tablespace  
LONG IN long-tablespace
```

The stored procedures in *syncdb2.sql* and *syncdb2long.sql* are implemented in Java in the files *SyncDB2.class* and *syncdb2long.class*. The source code is provided in *SyncDB2.java* and *SyncDB2long.java*. These scripts use the tilde character (~) as a command delimiter.

The default tablespace (usually called USERSPACE1) of a DB2 database that you wish to use as a consolidated database must use 8 kb pages.

☞ For an example using the sample application, see [“The CustDB Sample Application” on page 429](#).

Tips

☞ For tips on using IBM DB2 as a MobiLink consolidated database, see [“DBMS-dependent scripts” on page 65](#) and [“Supported DBMS scripting strategies” on page 65](#).

ODBC driver

You must set up an ODBC DSN for your DB2 UDB consolidated database.

For Solaris, Linux, and AIX, SQL Anywhere Studio includes an iAnywhere Solutions ODBC driver for IBM DB2.

For Windows, we recommend that you use the ODBC driver provided by IBM.

☞ You must configure the driver to work with MobiLink. For more information, see [“iAnywhere Solutions ODBC driver for DB2”](#) [*iAnywhere Solutions ODBC Drivers*, page 47].

Setting up a Microsoft SQL Server consolidated database

To set up Microsoft SQL Server to work as a MobiLink consolidated database, run the *syncmss.sql* setup script, located in the *MobiLink\setup* subdirectory of your SQL Anywhere installation.

ODBC driver

You must set up an ODBC DSN for your Microsoft SQL Server consolidated database. Unlike the other consolidated databases, iAnywhere Solutions does not provide an ODBC driver for Microsoft SQL Server. This is because the Microsoft SQL Server driver is freely available for download.

However, there are two changes that you should make to the default configuration to use Microsoft SQL Server as a consolidated database. In the Microsoft SQL Server DSN Configuration dialog, remove the check marks from the following options:

- ◆ Use ANSI quoted identifiers

- ◆ Use ANSI nulls, paddings and warnings

For updated details, see

http://www.ianymwhere.com/developer/technotes/odbc_mobilink.html.

The MobiLink synchronization server

All MobiLink clients synchronize through the MobiLink synchronization server. None connect directly to a database server. You must start the MobiLink synchronization server before asking a MobiLink client to synchronize.

Running the MobiLink synchronization server

The MobiLink synchronization server opens connections, via ODBC, with your consolidated database server. It then accepts connections from remote applications and controls the synchronization process.

❖ To start the MobiLink synchronization server

1. Run `dbmlsrv9`. Use the `-c` option to specify the ODBC connection parameters for your consolidated database.

☞ For information about connection parameters, see “`-c` option” [*MobiLink Synchronization Reference*, page 10].

You must specify connection parameters. Other options are available, but are optional. These options allow you to specify how the server works. For example, you can specify a maximum number of worker threads, cache size, and logging options.

☞ For more information about `dbmlsrv9` options, see “MobiLink Synchronization Server Options” [*MobiLink Synchronization Reference*, page 3].

Note: The `dbmlsrv9` options allow you to specify how the MobiLink synchronization server works. To control what the server does, you define scripts that are invoked at synchronization events.

☞ For more information, see “[MobiLink events](#)” on page 22.

Example

The following command starts the MobiLink synchronization server, identifying the ODBC data source *UltraLite 9.0 Sample* as the consolidated database. Enter the entire command on one line.

```
dbmlsrv9
-c "dsn=UltraLite 9.0 Sample;uid=DBA;pwd=SQL"
-zs MyServer
-o mlsrv.log
-vcr
-x tcpip
```

In this example, the `-zs` option provides a server name. The `-o` option specifies that the log file should be named *mlsrv.log*. The contents of

`mlsrv.log` are verbose because of the `-vcr` option. The `-x` option specifies that MobiLink clients will be permitted to connect via TCP/IP.

☞ You can also start the MobiLink synchronization server as a Windows service or UNIX daemon. For more information, see [“Running MobiLink Outside the Current Session”](#) on page 329.

Stopping the MobiLink synchronization server

The MobiLink synchronization server can be stopped from the computer where the server was started. You can stop the MobiLink server in the following ways:

- ◆ Click Shutdown on the MobiLink server window.
- ◆ Use Exit from the System tray context menu.
- ◆ Use the `dbmlstop` utility.

☞ For more information, see “MobiLink stop utility” [*MobiLink Synchronization Reference*, page 303].

Logging MobiLink synchronization server actions

Logging the actions that the server takes is particularly useful during the development process, and when troubleshooting. Verbose output is not recommended for normal operation of a production environment because it can slow performance.

Logging output to a file Logging output is sent to the MobiLink synchronization server window. In addition, you can send the output to a log file using the `-o` option. The following command sends output to a log file named `mlsrv.log`.

```
dbmlsrv9 -o mlsrv.log -c ...
```

☞ For more information, see “`-o` option” [*MobiLink Synchronization Reference*, page 13].

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

- ◆ With the `-on` option, you specify the size at which the log file is renamed with the extension `.old`, and a new file is started with the original name.
- ◆ With the `-os` option, you specify the size at which a new log file is started with a new name based on the date and a sequential number.
- ◆ With the `-ot` option, the contents of the log file are deleted before messages are sent to it.

☞ For more information, see

- ◆ “-on option” [*MobiLink Synchronization Reference*, page 14]
- ◆ “-os option” [*MobiLink Synchronization Reference*, page 15]
- ◆ “-ot option” [*MobiLink Synchronization Reference*, page 16]

Controlling the amount of logging output You can control the amount of output that is logged using the -v option.

☞ For more information, see “-v option” [*MobiLink Synchronization Reference*, page 21].

Controlling which errors are reported ☞ You can also control which warning messages are reported.

☞ For more information, see

- ◆ “-zw option” [*MobiLink Synchronization Reference*, page 31]
- ◆ “-zwd option” [*MobiLink Synchronization Reference*, page 32]
- ◆ “-zwe option” [*MobiLink Synchronization Reference*, page 33]

MobiLink clients

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

- ◆ Adaptive Server Anywhere
- ◆ UltraLite

Adaptive Server Anywhere clients

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

☞ For more information about Adaptive Server Anywhere clients, see [“Adaptive Server Anywhere Clients” on page 167](#).

UltraLite clients

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

The UltraLite application and libraries handle the synchronization actions at the application end. You can write your UltraLite application with little regard to synchronization. The UltraLite runtime keeps track of changes made since the previous synchronization.

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

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

☞ For more information about initiating synchronization, see [“Synchronization for UltraLite Applications”](#) [*UltraLite Database User’s Guide*, page 143].

☞ For more information about UltraLite clients, see [“UltraLite Clients” on page 207](#).

Specifying the communications protocol for clients

The MobiLink synchronization server has the `-x` command line option to specify the communications protocol or protocols for the synchronization client to connect to the MobiLink server. The kind of communication protocol you choose must match the synchronization protocol used by the client. The syntax for this command line option is:

dbmlsrv9 -c "connection-string" -x protocol

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

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

You can configure your protocol using communication parameters of the form:

(keyword=value;...)

For example:

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

☞ For more information about communication protocols, see [“-x option” \[MobiLink Synchronization Reference, page 24\]](#).

MobiLink users

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

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

☞ For more information about MobiLink users, see

- ◆ [“About MobiLink users” on page 104](#)
- ◆ [“Creating MobiLink users” on page 178](#)
- ◆ [“ml_user” \[MobiLink Synchronization Reference, page 320\]](#)

The synchronization process

A **synchronization** is the process of bidirectional data exchange between the MobiLink client and synchronization server. During this process, the client must establish and maintain a connection to 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 synchronization server.

The upload stream and the download stream

To upload rows, MobiLink clients prepare and send an **upload stream** 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 synchronization server prepares and sends a **download stream** that contains a list of inserts, updates, and deletes.

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

The upload stream consists of a set of new and old row values for each row modified in the remote database. 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 synchronization server receives the upload stream and applies the changes to the consolidated database. It normally applies all the changes in a single transaction. When it has finished, the MobiLink synchronization 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. MobiLink does so because repeatable reads are required for conflict detection purposes. If you have no conflict detection scripts or if you want to select an isolation level more suited to your needs, you can set this level in your `begin_connection` script.

2. **Download stream** The MobiLink synchronization server compiles a list of rows to be inserted, updated, or deleted on the MobiLink client, using synchronization logic that you create. It downloads these rows to

the MobiLink client. To compile this list, the MobiLink synchronization server opens a new transaction on the consolidated database.

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

Next, the MobiLink client automatically processes the download stream, 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.

3. **Optional download acknowledgement** The MobiLink client optionally sends a short confirmation message to the MobiLink synchronization server.

The MobiLink synchronization server receives the confirmation message. This message tells the synchronization server that the client has received and processed all downloaded changes. In response, it commits the download transaction begun in step 2.

☞ For more information about the SendDownloadAck extended option, see “SendDownloadACK (sa) extended option” [*MobiLink Synchronization Reference*, page 62] and “Send Download Acknowledgement synchronization parameter” [*UltraLite Database User’s Guide*, page 172].

During MobiLink synchronization, there are few distinct exchanges of information. The client builds and uploads the entire upload stream. In response, the synchronization server builds and downloads the entire download stream. 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.

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 your instructions outlining what you want done. The basic sequence is:

Event occurs ► Script is invoked (if it exists)

☞ For more information about synchronization events and scripts, see:

- ◆ “Synchronization Events” [*MobiLink Synchronization Reference*, page 83]
- ◆ [“Writing Synchronization Scripts” on page 37](#)

MobiLink scripts

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

Following are some typical synchronization scripts for tables.

Event	Script
upload_insert	<pre>INSERT INTO emp (emp_id,emp_name) VALUES (?,?)</pre>
upload_delete	<pre>DELETE FROM emp WHERE emp_id=?</pre>
upload_update	<pre>UPDATE emp SET emp_name=? WHERE emp_id=?</pre>
upload_old_row_insert	<pre>INSERT INTO old_emp (emp_id,emp_name) VALUES (?,?)</pre>
upload_new_row_insert	<pre>INSERT INTO new_emp (emp_id,emp_name) VALUES (?,?)</pre>
upload_fetch	<pre>SELECT id, name, size, quantity, unit_price FROM Product WHERE id=?</pre>

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` tables 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 emp_id, emp_name
FROM emp
WHERE emp_name = ?
```

The `download_cursor` acquires data from the `emp` table for a specified

emp_name.

COMMIT or ROLLBACK statements within scripts alter the transactional nature of the synchronization steps. If you use them, you 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.

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 synchronization server, to connect to a database, manipulate variables, and create user-defined procedures that can work with MobiLink and any supported relational database. There is a MobiLink Java API and a MobiLink .NET API that have routines to suit the needs of synchronization.

☞ For more information about programming synchronization logic, see [“Options for writing synchronization logic” on page 31](#).

☞ For information about DBMS-dependent scripting, such as scripting for Oracle, MS SQL Server, IBM DB2 or Adaptive Server Enterprise databases, see [“DBMS-dependent scripts” on page 65](#).

Storing scripts

SQL scripts are stored in system tables in the consolidated database. For Java and .NET, pointers to other locations are stored in the consolidated database. You can add all kinds of scripts to a consolidated database in two ways:

- ◆ By using stored procedures that are installed along with the MobiLink system tables when you create a consolidated database.
- ◆ By using Sybase Central.

☞ For more information, see [“Adding and deleting scripts in your consolidated database” on page 51](#).

Stored procedures

MobiLink stored procedures are used for programmatic conflict resolution, adding scripts, user authentication, and other customization procedures.

☞ For more information about using MobiLink stored procedures for customization, see [“Stored Procedures”](#) [*MobiLink Synchronization Reference*, page 261].

Other means to gain procedural control are commonly used with databases that don't have a defined procedural language. For example, with databases that do not permit user-defined procedures, such as IBM's DB2, Java procedures may be employed to act as MobiLink stored procedures.

☞ For more information about writing scripts using Java or .NET synchronization logic, see [“Writing Synchronization Scripts in Java”](#) on page 227 and [“Writing Synchronization Scripts in .NET”](#) on page 251.

For Adaptive Server Anywhere clients, you can use stored procedures called **client event hook procedures**, which are held on the remote database. A variety of event hook procedures are available for you to insert your own logic into the MobiLink synchronization process.

☞ For more information, see “Client event-hook procedures” [*MobiLink Synchronization Reference*, page 269].

Transactions in the synchronization process

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

The MobiLink synchronization server prepares the download stream, including all deletes, inserts, and updates, using another transaction. By default, it does not commit this transaction until it receives a positive confirmation from the MobiLink client. If the client confirms a successful download, the MobiLink synchronization server commits the download transaction. If the application encounters problems or cannot reply, the MobiLink synchronization server instead rolls back the download transaction.

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, you 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

The primary role of the download transaction is to select rows in the consolidated database. If the download fails being sent to the remote, the remote will upload the same timestamp over again, and no data will be lost.

The MobiLink synchronization 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 useful when analyzing failed synchronization attempts.

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

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. For example, in an embedded SQL UltraLite application, the SQLCode is set to SQLE_COMMUNICATION_ERROR when ULSynchronize returns.

There are three cases that are handled in different ways:

- ◆ **Failure during upload** If the failure occurs while building or applying the upload stream, the remote database is left in exactly the same state as at the start of synchronization. At the server, any part of the upload stream that has been applied will be rolled back.
- ◆ **Failure between upload and download** If the failure occurs once the upload stream is complete, but before the MobiLink client receives the download stream, the client cannot be certain whether the uploaded changes were successfully applied to the consolidated database. The upload stream might be fully applied and committed, or the failure may have occurred before the server applied the entire upload stream. The MobiLink synchronization 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 stream before building the new upload stream. If the previous upload was not committed, the new upload stream contains all changes from the previous upload stream.

- ◆ **Failure during download** If the failure occurs in the remote device while applying the download stream, 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. The MobiLink synchronization 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/user need not worry about maintaining consistent data in their application.

How the upload stream is processed

When the MobiLink synchronization server receives an upload stream from a MobiLink client, the entire upload stream is stored until the synchronization is complete. This is done for three purposes.

- ◆ **Deadlock** When an upload stream 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 synchronization 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 synchronization server automatically starts applying the upload stream from the beginning again.

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.

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

It would be difficult to write a `download_cursor` script that omits from the download stream rows that were sent as part of the upload. For this reason, the MobiLink synchronization server automatically removes these rows from the download stream. When a row is being added to the download stream, the MobiLink synchronization server locates the row in the upload stream and omits the row from the download stream when it is found to be the same.

- ◆ **Processing deletes after inserts and updates** The upload stream is applied to the consolidated database in an order that avoids referential integrity violations. The upload stream is formatted so all operations (inserts, updates, and deletes) for a single table are grouped together. The tables in the upload stream are ordered based on foreign key relationships. All tables in the remote database that are referenced by another table in the remote database will be in the upload stream before the referencing table.

For example, if table A and table C both have foreign keys that reference a primary key column in B, then table B rows are uploaded first.

When the upload stream is applied to the consolidated database, the inserts and updates are applied in the order they appear in the upload stream. When an inserted or updated row references a newly inserted row, this ensures the referenced row will be inserted before the referencing row. Deletes are applied in the opposite order after all inserts and updates have been applied. When a row being deleted references another row that is also being deleted, this order of operations ensures the referencing row is deleted before the referenced row is deleted.

Referential integrity and synchronization

All MobiLink clients enforce referential integrity when they incorporate the download stream into the remote database.

Rather than failing the download transaction, 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.

Referential integrity checked at the end of the transaction

The MobiLink client incorporates changes from the download stream 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.

Errors are avoided

The MobiLink client resolves referential integrity violations automatically. This feature minimizes administration requirements. It also prevents an error in a synchronization script from disabling an MobiLink client.

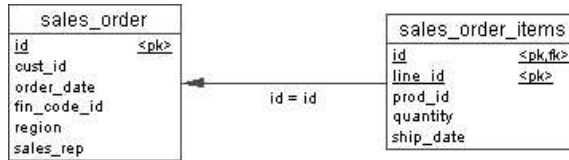
An efficient way to delete rows

You can exploit the automatic referential integrity mechanism of MobiLink clients to delete large quantities of information in a very efficient manner. If your MobiLink client contains a primary row, and other rows that reference

it, you can delete all the referencing rows simply by synchronizing a delete of the primary row.

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 automatic 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 will be 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 stream is now smaller. This technique is particularly valuable when using slow communication methods.

Upload-only and download-only synchronization

Adaptive Server Anywhere remote databases

For both Adaptive Server Anywhere and UltraLite remote databases, you can choose to do a full synchronization, or you can perform only an upload or download.

In Adaptive Server Anywhere remote databases, you perform upload-only synchronization using the dbmlsync option `-ds` or the extended option `DownloadOnly`.

☞ For more information, see “`-ds` option” [*MobiLink Synchronization Reference*, page 43] and “`DownloadOnly (ds)` extended option” [*MobiLink Synchronization Reference*, page 49].

In Adaptive Server Anywhere remote databases, you perform download-only synchronization using the dbmlsync option `-uo` or the extended option `UploadOnly`.

☞ For more information, see “`-uo` option” [*MobiLink Synchronization Reference*, page 80] or “`UploadOnly (uo)` extended option” [*MobiLink Synchronization Reference*, page 65].

UltraLite remote databases

In UltraLite remote databases, you perform download-only synchronization using the `Download Only` synchronization parameter.

☞ For more information, see “`Download Only` synchronization parameter” [*UltraLite Database User's Guide*, page 165].

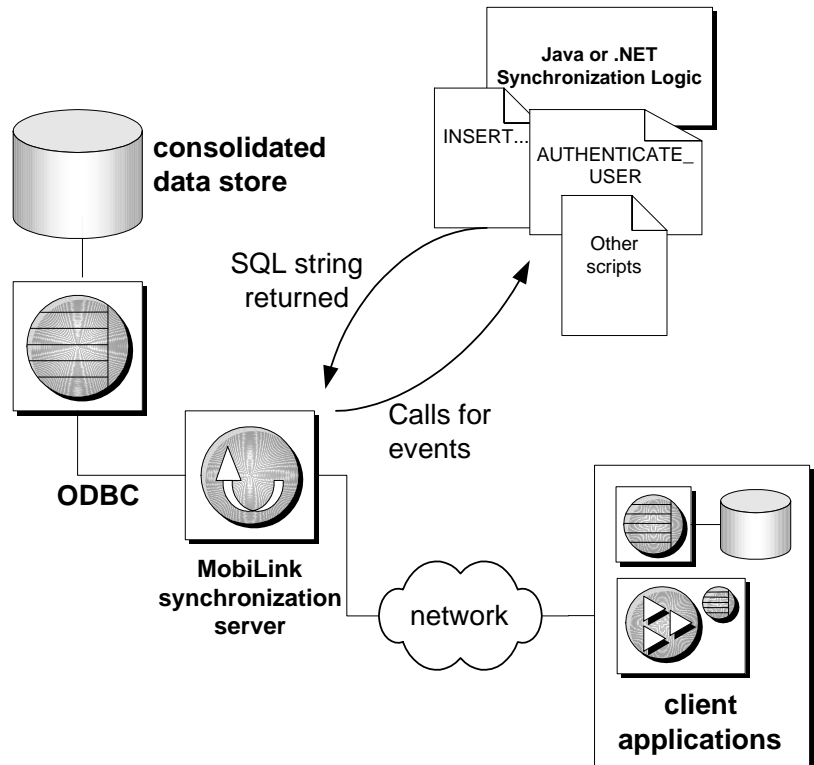
In UltraLite remote databases, you perform upload-only synchronization using the `Upload Only` synchronization parameter.

☞ For more information, see “`Upload Only` synchronization parameter” [*UltraLite Database User's Guide*, page 176].

Options for writing synchronization logic

MobiLink synchronization scripts can be written in SQL, in Java, or in .NET programming languages. Java or .NET are a good choice whenever your design is restricted by the limitations of the SQL language or by the capabilities of your database-management system, or if you want DBMS-independent synchronization logic.

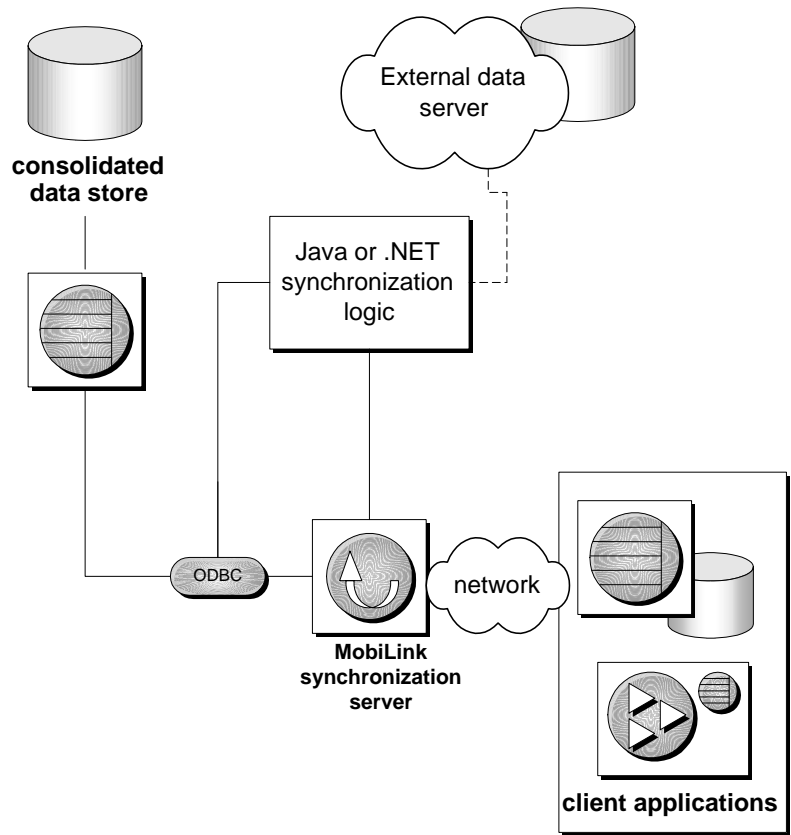
Program synchronization logic can function just as SQL logic functions, as shown in the figure below. The MobiLink synchronization server can make calls to Java or .NET methods on the occurrence of MobiLink events just as it can access SQL scripts on the occurrence of MobiLink events. However, the upload and download streams are not directly accessible from Java or .NET synchronization logic, where a SQL string must be returned to MobiLink.



SQL synchronization logic is restricted to the procedural language capabilities of your consolidated database. SQL languages are unlikely to offer all the programming logic given by Java or .NET programming languages. You might want to use Java or .NET synchronization logic when

your SQL logic is limited, when you need to perform operations across database platforms, and when you need portability across RDBMSs and operating systems. Following are some scenarios where you might want to consider writing scripts in Java or .NET.

- ◆ A user authentication procedure can be written in Java or .NET that inserts the user ID of a MobiLink user into a table on the consolidated database for audit purposes.
- ◆ 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.
- ◆ If your database lacks the ability to make user-defined stored procedures, you can make a method in Java or .NET that can perform the needed functionality.
- ◆ 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.
- ◆ With Java and .NET synchronization logic, you can use MobiLink to access data from application servers, Web servers, and files. You can use JDBC or iAnywhere classes in your synchronization logic to access data in relational databases other than the consolidated database. For example, an external server can be used to validate a user ID and password. The figure below shows the links between Java or .NET synchronization logic and both a consolidated database and a second data server.



MobiLink APIs

With Java and .NET synchronization logic, you have access to a MobiLink API. The MobiLink APIs are sets of classes and interfaces for MobiLink synchronization. There are two MobiLink APIs: Java and .NET.

The MobiLink Java API offers you:

- ◆ Access to the existing ODBC connection as a JDBC connection.
- ◆ The ability to create new JDBC connections to perform commits or connects outside the current synchronization connection. For example, you can use this for error logging.
- ◆ 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.

-
- ◆ Code that runs inside the Java virtual machine and allows access to all Java libraries and Java Native Interface calls.

☞ For more information, see [“MobiLink Java API Reference” on page 246](#).

The MobiLink .NET API offers you:

- ◆ Access to the existing ODBC connection using iAnywhere classes that call ODBC from .NET.
- ◆ Code that runs inside the .NET Common Language Runtime (CLR) and allows access to all .NET libraries and unmanaged calls.

☞ For more information, see [“MobiLink .NET API Reference” on page 269](#).

Further reading

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

- ◆ [“Writing Synchronization Scripts” on page 37](#)
- ◆ [“Synchronization Techniques” on page 69](#)
- ◆ [“Writing Synchronization Scripts in Java” on page 227](#)
- ◆ [“Writing Synchronization Scripts in .NET” on page 251](#)

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 an Adaptive Server Anywhere consolidated database, see “Keeping Your Data Secure” [*SQL Anywhere Studio Security Guide*, page 3].

- ◆ **Protecting data in the remote databases** If you are using Adaptive Server Anywhere remote databases, the data can be protected using Adaptive Server 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 an Adaptive Server Anywhere remote database, see “Keeping Your Data Secure” [*SQL Anywhere Studio Security Guide*, page 3].

- ◆ **Protecting data during synchronization** Communication from MobiLink clients to MobiLink synchronization servers can be protected by the MobiLink transport layer security features.

☞ For more information, see [“Transport-Layer Security” on page 337](#).

- ◆ **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 [“Authenticating MobiLink Users” on page 103](#).

CHAPTER 3

Writing Synchronization Scripts

About this chapter

You control the synchronization process by writing synchronization scripts and storing them in the consolidated database.

You can write scripts in SQL, Java, or .NET. This chapter applies to all kinds of scripts, but focuses on how to write synchronization scripts in SQL.

For more information about writing scripts, see

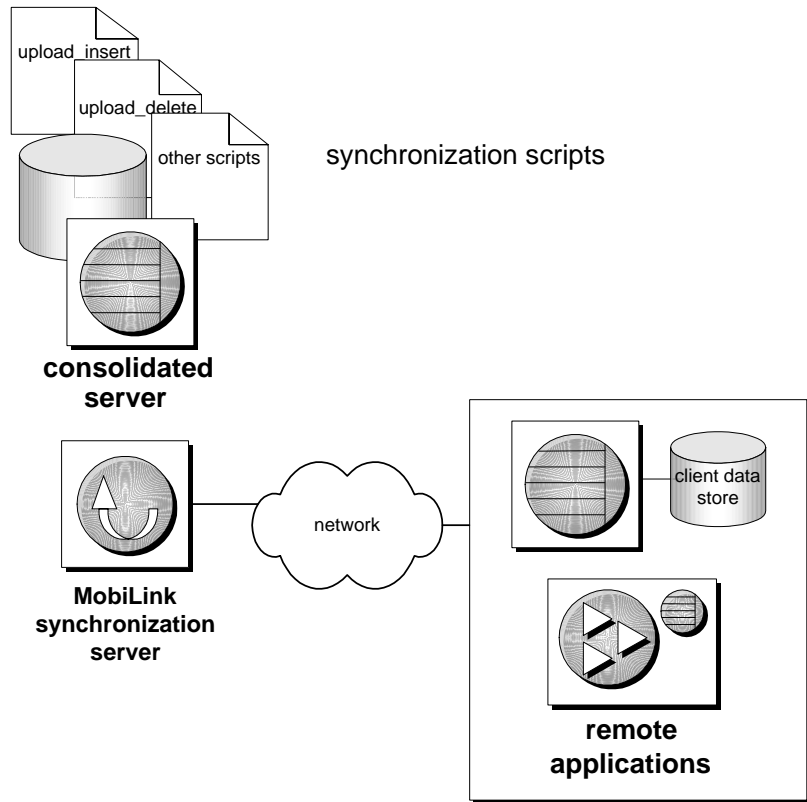
- ◆ “Synchronization Events” [*MobiLink Synchronization Reference*, page 83]
- ◆ “Synchronization Techniques” on page 69
- ◆ “Writing Synchronization Scripts in Java” on page 227
- ◆ “Writing Synchronization Scripts in .NET” on page 251

Contents

Topic:	page
Introduction to synchronization scripts	38
Scripts and the synchronization process	44
Script types	46
Script parameters	48
Script versions	49
Adding and deleting scripts in your consolidated database	51
Writing scripts to upload rows	54
Writing scripts to download rows	56
Writing scripts to handle errors	62
Testing script syntax	64
DBMS-dependent scripts	65

Introduction to synchronization scripts

MobiLink Synchronization logic consists of scripts, which may be individual statements or stored procedure calls, stored in your consolidated database. During synchronization, the MobiLink synchronization server reads the scripts and executes them against the consolidated database. Scripts provide you with opportunities to perform tasks at various points of time during the synchronization process. You can use Sybase Central to add scripts to the consolidated database or you can use stored procedures.



The synchronization process is composed of multiple steps. A unique **event name** identifies each step. You control the synchronization process by writing scripts associated with some of these events. You write a script only when some particular action must occur at a particular event. The MobiLink synchronization server executes each script when its associated event occurs. If you do not define a script for a particular event, the MobiLink synchronization server simply proceeds to the next step.

For example, one event is `begin_upload_rows`. You can write a script and associate it with this event. The MobiLink synchronization server reads this script when it is first needed, and executes it during the upload phase of synchronization. If you write no script, the MobiLink synchronization server proceeds immediately to the next step, which is processing the uploaded rows.

Some scripts, called **table scripts**, are associated not only with an event, but also with a particular table in the remote database. The MobiLink synchronization server performs some tasks on a table-by-table basis; for example, downloading rows. You can have many scripts associated with the same event, but each with different application tables. Alternatively, you can define many scripts for some application tables, but none for others.

☞ For an overview of events, see [“The synchronization process” on page 21](#).

☞ For a description of every script you can write, see “Synchronization Events” [*MobiLink Synchronization Reference*, page 83].

You can write scripts in SQL, Java, or .NET. This chapter applies to all kinds of scripts, but focuses on how to write synchronization scripts in SQL.

☞ For a description and comparison of SQL, Java, and .NET, see [“Options for writing synchronization logic” on page 31](#).

☞ For information about writing scripts in .NET, see [“Writing Synchronization Scripts in .NET” on page 251](#).

☞ For information about writing scripts in Java, see [“Writing Synchronization Scripts in Java” on page 227](#).

☞ For information about how to implement synchronization scripts, see [“Synchronization Techniques” on page 69](#).

A simple synchronization script

MobiLink provides many events that you can exploit, but it is not mandatory to provide scripts for each event. In a simple synchronization model, you may need only a few scripts.

Downloading all the rows from the table to each remote database synchronizes the `ULProduct` table in the `CustDB` sample application. In this case, no additions are permitted at the remote databases. You can implement this simple form of synchronization with a single script; in this case only one event has a script associated with it.

The MobiLink event that controls the rows to be downloaded during each

synchronization is named the `download_cursor` event. Cursor scripts must contain `SELECT` statements. The MobiLink synchronization server uses these queries to define a cursor. In the case of a `download_cursor` script, the cursor selects the rows to be downloaded to one particular table in the remote database.

In the CustDB sample application, there is a single `download_cursor` script for the `ULProduct` table in the sample application, which consists of the following query.

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

This query generates a result set. The rows that make up this result set are downloaded to the client. In this case, all the rows of the table are downloaded.

The MobiLink synchronization server knows to send the rows to the `ULProduct` application table because this script is associated with both the `download_cursor` event and `ULProduct` table by the way it is stored in the consolidated database. Sybase Central allows you to make these associations.

Note

In this example, the query selects data from a consolidated table also named `ULProduct`. The names need not match. You could, instead, download data to the `ULProduct` application table from any table, or any combination of tables, in the consolidated database by rewriting the query.

You can write more complicated synchronization scripts. For example, you could write a script that downloads only recently modified rows, or one that provides different information to each remote database.

Generating scripts automatically

You can use the `dbmlsrv9 -za` option to generate default synchronization scripts. The synchronization scripts perform a snapshot synchronization of your consolidated database with your remote database using table and column names that are sent from the client.

To use this feature with Adaptive Server Anywhere clients, set the `SendColumnNames` extended option to `ON` to cause `dbmlsync` to send the column names with the upload header. To use this feature with UltraLite clients, set the `send_column_names` parameter to `ul_true`.

When you use `-za`, scripts are generated the first time that a remote synchronizes with a script version that doesn't exist. If the given script

version already exists, `-za` has no effect. This means that you cannot use `-za` to generate scripts one table at a time for the same script version. Using `-za`, you must generate scripts for all tables and publications at once.

☞ For more information, see “`-za` option” [*MobiLink Synchronization Reference*, page 28].

Example

Start the MobiLink synchronization server using the `-za` switch. For example, type:

```
dbmlsrv9 -c "dsn=YourDBDSN" -za
```

Run `dbmlsync` and set the `SendColumnNames` extended option to ON. For example, type:

```
dbmlsync -c dsn=dsn_remote -e "SendColumnNames=ON"
```

Scripts are generated for all tables specified in the publication. On synchronization, these automatically-generated scripts control the upload and download of data to and from your client and consolidated databases. The following table describes these scripts for the `emp` table.

Script name	Script contents
upload_insert	INSERT INTO emp (emp_id, emp_name) VALUES (?,?)
upload_update	UPDATE emp SET emp_name=? WHERE emp_id=?
upload_delete	DELETE FROM emp WHERE emp_id=?
download_cursor	SELECT emp_id, emp_name FROM emp

Generating example scripts

You can use the `dbmlrv9 -ze` option to generate example synchronization scripts. The example synchronization scripts are capable of performing a snapshot synchronization of your consolidated database with your remote database using the table and column names sent from the client, but they are not enabled. If the consolidated database has different table or column names, then activating these scripts causes an error during the synchronization.

To use this feature with Adaptive Server Anywhere clients, set the

SendColumnNames extended option to ON to cause dbmlsync to send the column names with the upload header. To use this feature with UltraLite clients, set the send_column_names parameter to ul_true.

The -ze option generates the example scripts example_upload_insert, example_upload_update, example_upload_delete, and example_download_cursor.

☞ For more information, see “-ze option” [*MobiLink Synchronization Reference*, page 29].

Example

The following example generates scripts for an Adaptive Server Anywhere remote database.

At a command prompt, type:

```
dbmlsrv9 -c "dsn=YourDBDSN" -ze
```

At a command prompt, type:

```
dbmlsync -c dsn=dsn_remote -e "SendColumnNames=ON"
```

In the example above, example scripts are generated for all tables specified in the synchronization definition. The scripts exist for each table specified in the synchronization definition. The following table lists these scripts for the emp table.

Script name	Script
example_upload_insert	INSERT INTO emp (emp_id,emp_name) VALUES (?,?)
example_upload_update	UPDATE emp SET emp_name=? WHERE emp_id=?
example_upload_delete	DELETE FROM emp WHERE emp_id=?
example_download_cursor	SELECT emp_id, emp_name FROM emp

The example scripts select and upload all records from any table in the synchronization subscription that meet the conditions specified in the statement. So, for example, the upload_insert script for emp inserts all records from emp. The example scripts are generated for each table in the remote database specified in the synchronization subscription. The MobiLink synchronization server generates complete scripts needed for a

snapshot synchronization. The scripts are added right after the synchronization description is processed. The synchronization is aborted after scripts are generated.

Example scripts for UltraLite

When you build an UltraLite application, the UltraLite generator automatically inserts an `example_upload_cursor` script and an `example_download_cursor` script into the UltraLite reference database. The action of the example download script is to download all rows of a corresponding table that exists in the remote database. These scripts specify the select list in the correct order, and the `example_upload_cursor` script also includes the correct `WHERE` clause.

The example scripts are inserted into the `ml_scripts` table, but they are not used unless you insert an entry in the `ml_table_script` table that associates them with the `upload_cursor` or `download_cursor` event, respectively.


Minimally, the example scripts for download cursors provide the order of columns expected by the remote database.


Scripts and the synchronization process


Each script corresponds to a particular event in the synchronization process. You write a script only when some action must occur. All unnecessary events can be left undefined.

The two principal parts of the process are the processing of uploaded information and the preparation of rows for downloading.

The MobiLink synchronization server reads and prepares each script once, when it is first needed. The script is then executed whenever the event is invoked.

The sequence of events  For information about the full sequence of MobiLink events, see “Overview of MobiLink events” [*MobiLink Synchronization Reference*, page 86].

 For the details of upload stream processing, see [“Writing scripts to upload rows” on page 54](#).

 For the details of download stream processing, see [“Writing scripts to download rows” on page 56](#).

Notes

- ◆ MobiLink technology allows multiple clients to synchronize concurrently. In this case, each client uses a separate connection to the consolidated database.
- ◆ The `begin_connection` and `end_connection` events are independent of any one synchronization as one connection can handle many synchronization requests. These scripts have no parameters. These are examples of connection-level scripts.
- ◆ Some events are invoked only once for each synchronization and have a single parameter. This parameter is the user name, which uniquely identifies the MobiLink client that is synchronizing. These are also examples of connection-level scripts.
- ◆ Some events are invoked once for each table being synchronized. Scripts associated with these events are called table-level scripts. They provide two parameters. The first is the user name supplied in the call to the synchronization function, and the second is the name of the table in the remote database being synchronized.

While each table can have its own table scripts, you can also write table-level scripts that are shared by several tables.
- ◆ Some events, such as `begin_synchronization`, occur at both the connection level and the table level. You can supply both connection and table scripts for these events.

- ◆ The COMMIT statements illustrate how the synchronization process is broken up into distinct transactions.
- ◆ Errors are a separate event that can occur at any point within the synchronization process. Errors are handled using the following script.

```
handle_error( error_code, error_message, user_name, table_
              name )
```

☞ For reference material, including detailed information about each script and its parameters, see “Synchronization Events” [*MobiLink Synchronization Reference*, page 83].

Script types

Synchronization scripts can apply to the entire connection or to specific tables.

- ◆ **connection scripts** These scripts perform actions that are connection-specific or synchronization-specific and that are independent of any one remote table. These scripts are used in conjunction with other scripts when implementing more complex synchronization schemes.
- ◆ **table scripts** These scripts perform actions specific to one synchronization and one particular remote table. These scripts are used in conjunction with other scripts when implementing more complex synchronization schemes.

Connection scripts

Connection-level scripts control high level events that are not associated with a particular table. Use these events to perform global tasks that are required during every synchronization.

Connection scripts control actions centered on connecting and disconnecting, as well as actions at synchronization-level events such as beginning and ending the upload or download process. Some connection scripts have related table scripts. These connection scripts are always invoked regardless of the tables being synchronized.

You only need to write a connection-level script when some action must occur at a particular event. You may need to create scripts for only a few events. The default action at any event is for the MobiLink synchronization server to carry out no actions. Some simple synchronization schemes need no connection scripts.

Table scripts

Table scripts allow actions at specific events relating to the synchronization of a specific table, such as the start or end of uploading rows, resolving conflicts, or selecting rows to download.

The synchronization scripts for a given table can refer to any table, or combination of tables, in the consolidated database. You can use this feature to fill a particular remote table with data stored in one or more consolidated tables, or to store data uploaded from a single remote table into multiple tables in the consolidated database.

Table names need not match

The names of tables in the remote databases need not match the names of the tables in the consolidated database. The MobiLink synchronization server

determines which scripts are associated with a table by looking up the remote table name in the `ml_table` system table.

Script parameters

Most synchronization scripts receive parameters from the MobiLink synchronization server. You can use these parameters in your scripts by placing question marks in the script.

The following are some common parameters used in scripts.

- ◆ **last_download_timestamp** The last download timestamp is the value obtained from the consolidated database during the last successful synchronization immediately prior to the download phase. If the current MobiLink user has never synchronized, or has never synchronized successfully, this value is set to 1900-01-01.

☞ For more information, see [“Timestamp-based synchronization” on page 72](#).

- ◆ **ml_username** The value of this parameter is the string that uniquely identifies a MobiLink client. Each client must identify itself by this name when initiating synchronization with a MobiLink synchronization server. This parameter is available within most connection-level scripts, all table-level scripts, and some cursor scripts.

The user name can be used to partition tables among remote databases.

- ◆ **table** This parameter identifies a table in the remote database. The consolidated database may or may not contain a table with the same name. Only table scripts use this parameter.

To use parameters, place a single question mark in your SQL script for each parameter. Some parameters are optional. The MobiLink synchronization server replaces each question mark with the value of a parameter. It substitutes values in the order the parameters appear in the script definition.

☞ For reference material, including detailed information about each script and its parameters, see “Synchronization Events” [*MobiLink Synchronization Reference*, page 83].

Script versions

Scripts are organized into groups called **script versions**. By specifying a particular version, MobiLink clients can select which set of synchronization scripts will be used to process the upload stream and prepare the download stream.

☞ For information about how to add a script version to the consolidated database, see [“Adding a script version” on page 50](#).

Application of script versions

Script versions allow you to organize your scripts into sets, which are run under different circumstances. This ability provides flexibility and is especially useful in the following circumstances.

- ◆ **customization** Using a different set of scripts to process information from different types of remote users. For example, you could write a different set of scripts for use when managers synchronize their databases than would be used for other people in the organization. Although you could achieve the same functionality with one set of scripts, these scripts would be more complicated.
- ◆ **upgrading applications** When you wish to upgrade a database application, new scripts may be needed because the new version of your application may handle data differently. New scripts are almost always necessary when the remote database changes. It is usually impossible to upgrade all users simultaneously. MobiLink clients can request that a new set of scripts be used during synchronization. Since both old and new scripts can coexist on the server, all users can synchronize no matter which version of your application they are using.
- ◆ **multiple applications** A single MobiLink synchronization server may need to synchronize two entirely different applications. For example, some employees may use a sales application, whereas others require an application designed for inventory control. When two applications require different sets of data, you can create two versions of the synchronization scripts, one version for each application.

Assigning version names


A script version name is a string. You specify this name when you add a script to the consolidated database. For example, if you add your scripts with the `ml_add_connection_script` and the `ml_add_table_script` stored procedures, the script version name is the first parameter. Alternatively, if you add your scripts using Sybase Central, you are prompted for the version name.

The default script version

Whenever a remote site fails to supply a script version, the MobiLink synchronization server uses the first version defined in the `ml_script_version` table. If no script version has been defined, the synchronization fails.

Adding a script version

All scripts are associated with a script version. You must add a version name to your consolidated database before you can add any connection scripts.

 For more information, see [“Script versions” on page 49](#).

❖ To add a script version to a database (Sybase Central)


1. From Sybase Central, right-click MobiLink Synchronization and connect to the consolidated database.
2. Open the Versions folder.
3. Double-click Add Version and follow the instructions in the wizard.

❖ To remove a script version from a database (Sybase Central)

1. From Sybase Central, right-click MobiLink Synchronization and connect to the consolidated database.
2. Open the Versions folder.
3. Right-click the version name and select Delete.
4. The Confirm Delete dialog appears. Click Yes.

❖ To add a script version to a database (stored procedures)

1. You can add a script version in the same operation as adding a connection script or table script.

 For more information, see “Stored procedures to add or delete scripts” [*MobiLink Synchronization Reference*, page 262].

Adding and deleting scripts in your consolidated database

When you have created scripts, you must add them to MobiLink system tables in the consolidated database. To do this, you can use stored procedures or Sybase Central wizards.

☞ For information about the MobiLink system tables, see “MobiLink System Tables” [*MobiLink Synchronization Reference*, page 315].

Note: SQL scripts are stored in the consolidated database. In the case of Java and .NET scripts, the location of the script is stored in the consolidated database. However, the method for adding/deleting a script or script location is similar.

Adding or deleting scripts

You can add synchronization scripts using Sybase Central wizards. The procedure is different for connection scripts and table scripts. Table scripts correspond to tables in the remote database, so before you can add a table script, you must add the name of the remote database table to the consolidated database.

If you are using Sybase Central, you must add a synchronization version to the database before you can add individual scripts. For more information, see [“Adding a script version” on page 50](#).

❖ To add or delete a connection script (Sybase Central)

1. From Sybase Central, right-click MobiLink Synchronization and connect to the consolidated database.
2. Open Connection Scripts.
3. To add a connection script, double-click Add Connection Script and follow the instructions in the wizard.

or

To delete a connection script, right-click the script name and select Delete. The Confirm Delete dialog appears. Click Yes.

❖ **To add or delete a remote table in the list of synchronized tables (Sybase Central)**

1. From Sybase Central, right-click MobiLink Synchronization and connect to the consolidated database.
2. Open Synchronized Tables.
3. To add a remote table to the list of synchronized tables, double-click Add Synchronized Table. Enter the name of a table at the remote database for which you are going to write synchronization scripts. The wizard provides a shortcut if the consolidated database has a table with a matching name.

or

To delete a remote table from the list of synchronized tables, right-click the table name and select Delete. The Confirm Delete dialog appears. Click Yes.

❖ **To add or delete a table script in a database (Sybase Central)**

1. From Sybase Central, right-click MobiLink Synchronization and connect to the consolidated database.
2. Open Synchronized Tables.
3. Select the table for which you wish to add a script.
4. To add a table script, double-click Add Table Script and follow the instructions in the wizard.

or

To delete a table script, right-click the script name and select Delete. The Confirm Delete dialog appears. Click Yes.

❖ **To add or delete all types of scripts (stored procedures)**

1. You can add scripts to a consolidated database or delete scripts from a consolidated database using stored procedures that are installed along with the MobiLink system tables when you create your consolidated database.

☞ For a description of the stored procedures that you can use to add or delete scripts, see “Stored procedures to add or delete scripts” [*MobiLink Synchronization Reference*, page 262].

Direct inserts of scripts

In most cases, it is recommended that you use stored procedures or Sybase Central to insert scripts into the system tables. However, in some rare cases you may need to use an INSERT statement to directly insert the scripts. For example, older versions of some DBMSs may have length limitations that make it difficult to use stored procedures.

☞ For a complete description of the MobiLink system tables, see “MobiLink System Tables” [*MobiLink Synchronization Reference*, page 315].

The format of the INSERT statements that are required to directly insert scripts can be found in the source code for the ml_add_connection_script and ml_add_table_script stored procedures. The source code for these stored procedures is located in the MobiLink setup scripts. There is a different setup script for each supported RDBMS. The setup scripts are:

Consolidated database	Setup file
Adaptive Server Anywhere	<i>scripts\syncasa.sql</i>
Oracle	<i>MobiLink\setup\syncora.sql</i>
IBM DB2 version 6 and later	<i>MobiLink\setup\syncdb2long.sql</i>
IBM DB2 prior to version 6	<i>MobiLink\setup\syncdb2.sql</i>
Microsoft SQL Server	<i>MobiLink\setup\syncmss.sql</i>
Adaptive Server Enterprise version 12.5 and later	<i>MobiLink\setup\syncase125.sql</i>
Adaptive Server Enterprise prior to version 12.5	<i>MobiLink\setup\syncase.sql</i>

Note: IBM DB2 prior to version 6 only supports column names and other identifiers of 18 characters or less, and so the names are truncated. For example, ml_add_connection_script is shortened to ml_add_connection_.

Writing scripts to upload rows

To upload information contained in your remote database to your consolidated database, you define upload scripts. You write separate scripts to handle rows that are updated, inserted, or deleted at the remote database. A simple implementation would carry out corresponding actions (update, insert, delete) at the consolidated database.

The MobiLink synchronization server uploads data in a single transaction. For a description of the upload process, see “Events during upload”

[*MobiLink Synchronization Reference*, page 92].

Notes

- ◆ The begin_upload and end_upload scripts for each remote table hold logic that is independent of the individual rows being updated.
- ◆ The upload stream consists of single row inserts, updates, and deletes. These actions are typically performed using upload_insert, upload_update and upload_delete scripts.
- ◆ To prepare the upload for Adaptive Server Anywhere clients, the dbmlsync utility requires access to all transaction logs written since the last successful synchronization. For more information, see “[Transaction log files](#)” on page 187.

Writing upload_insert scripts

The MobiLink synchronization server uses this event during processing of the upload stream to handle rows inserted into the remote database. The following INSERT statement shows how you use the upload_insert statement.

```
INSERT INTO emp (emp_id,emp_name)
VALUES (?,?)
```

☞ For more information, see “upload_insert table event” [*MobiLink Synchronization Reference*, page 218].

Writing upload_update scripts

The MobiLink synchronization server uses this event during processing of the upload stream to handle rows updated at the remote database. The following UPDATE statement illustrates use of the upload_update statement.

```
UPDATE emp
SET emp_name=?
WHERE emp_id=?
```

For more information, see “upload_update table event” [*MobiLink Synchronization Reference*, page 231].

Writing upload_delete scripts

The MobiLink synchronization server uses this event during processing of the upload stream to handle rows deleted from the remote database. The following statement shows how to use the upload_delete statement.

```
DELETE FROM emp
WHERE emp_id=?
```

For more information, see “upload_delete table event” [*MobiLink Synchronization Reference*, page 214].

Writing upload_fetch scripts

The upload_fetch script is a SELECT statement that defines a cursor in the consolidated database table. This cursor is used to compare the old values of updated rows, as received from the remote database, against the value in the consolidated database. In this way, the upload_fetch script identifies conflicts when updates are being processed.

Given a synchronized table defined as:

```
CREATE TABLE uf_example (
  pk1 integer NOT NULL,
  pk2   integer NOT NULL,
  val   varchar(200),
  PRIMARY KEY( pk1, pk2 ))
```

Then one possible upload_fetch script for this table is:

```
SELECT pk1, pk2, val
FROM uf_example
WHERE pk1 = ? and pk2 = ?
```

☞ For more information, see “upload_fetch table event” [*MobiLink Synchronization Reference*, page 216].

The MobiLink synchronization server requires the WHERE clause of the query in the upload_fetch script to identify exactly one row in the consolidated database to be checked for conflicts.

Writing scripts to download rows

There are two scripts that can be used for processing each table during the download transaction. These are the `download_cursor` script, which carries out inserts and updates, and the `download_delete_cursor` script, which carries out deletes.

These scripts are either `SELECT` statements or calls to procedures that return result sets. The MobiLink synchronization server downloads the result set of the script to the remote database. The MobiLink client automatically inserts or updates rows based on the `download_cursor` script result set, and deletes rows based on the `download_delete_cursor` event.

☞ For more information about using stored procedures, see [“Downloading a result set from a stored procedure call” on page 97](#).

The MobiLink synchronization server downloads data in a single transaction. For a description of the download process, see “Events during download” [*MobiLink Synchronization Reference*, page 96].

Notes

- ◆ Like the upload stream, the download stream starts and ends with connection events. Other events are table-level events.
- ◆ By default, if no confirmation of the download is received from the client, the entire download transaction is rolled back in the consolidated database. You can change this behavior with the `dbmsync` option “SendDownloadACK (sa) extended option” [*MobiLink Synchronization Reference*, page 62] or UltraLite “Send Download Acknowledgement synchronization parameter” [*UltraLite Database User’s Guide*, page 172].
- ◆ The `begin_download` and `end_download` scripts for each remote table hold logic that is independent of the individual rows being updated.
- ◆ The download stream does not distinguish between inserts and updates. The script associated with the `download_cursor` event is a `SELECT` statement that defines the rows to be downloaded. The client detects whether the row exists or not and carries out the appropriate insert or update operation.
- ◆ At the end of the download processing, the client automatically deletes rows if necessary to avoid referential integrity violations.

☞ For more information, see [“Referential integrity and synchronization” on page 28](#).

Writing `download_cursor` scripts

You write `download_cursor` scripts to download information from the

consolidated database to your remote database. You must write one of these scripts for each table in the remote database for which you want to download changes. You can use other scripts to customize the download process, but no others are necessary.

- ◆ Each `download_cursor` script must contain a `SELECT` statement or a call to a procedure that contains a `SELECT` statement. The MobiLink synchronization server uses this statement to define a cursor in the consolidated database.
- ◆ The script must select all columns that correspond to the columns in the corresponding table in the remote database. The columns in the consolidated database can have different names than the corresponding columns in the remote database, but they must be of compatible types.
- ◆ The columns must be selected in the order that the corresponding columns are defined in the remote database. This order is identical to the order of the columns in the reference database.

Example

The following script could serve as a `download_cursor` script for a remote table that holds employee information. The MobiLink synchronization server would use this SQL statement to define the download cursor. This script downloads information about all the employees.

```
SELECT emp_id, emp_fname, emp_lname
FROM employee
```

The MobiLink synchronization server passes specific parameters to some scripts. To use these parameters, you include a question mark in your SQL statement. The MobiLink synchronization server substitutes the value of the parameter before executing the statement against the consolidated database. The following script shows how you can use these parameters:

```
call ml_add_table_script( 'Lab', 'ULOrder', 'download_cursor',
'SELECT o.order_id, o.cust_id, o.prod_id, o.emp_id, o.disc,
    o.quant, o.notes, o.status
FROM ULOrder o
WHERE o.last_modified >= ?
AND o.emp_name = ?' )
```

In this example, the MobiLink synchronization server replaces the question mark with the value of the parameter to the `download_cursor` script.

Notes

- ◆ Row values can be selected from a single table or from a join of multiple tables.
- ◆ The script itself need not include the name of the remote table. The remote table need not have the same name as the table in the consolidated database. The name of the remote table is identified by an entry in the

ml_table table. In Sybase Central, the remote tables are listed together with their scripts.

- ◆ The rows in the remote table must contain the values of *emp_id*, *emp_fname*, and *emp_lname*. The remote columns must be in that order, although they can have different names. The columns in the remote database are in the same order as those in the reference database.

UltraLite tip

The example scripts list the columns in the order that they are defined in the reference database. Inspect the *example_download_cursor* and *example_upload_cursor* scripts to see the column order.

- ◆ All cursor scripts must select the columns in the same order as the columns are defined in the remote database. Where column names or table structure is different in the consolidated database, columns should be selected in the correct order for the remote database, or equivalently, the reference database. Columns are assigned to columns in the remote database based on their order in the *SELECT* statement.
- ◆ When you build an UltraLite application, the UltraLite generator creates a sample download script for each table in your UltraLite application. It inserts these sample scripts into your reference database. The example scripts assume that the consolidated database contains the same tables as your application. You must modify the sample scripts if your consolidated database differs in design, but these scripts provide a starting point.

Writing download_delete_cursor scripts

You write *download_delete_cursor* scripts to delete rows from your remote database. You must write one of these scripts for each table in the remote database from which you want to delete rows during synchronization.

You cannot just delete rows from the consolidated database and have them disappear from remote databases. You need to keep track of the primary keys for deleted rows, so that you can select those primary keys with your *download_delete_cursor*. There are two common techniques for achieving this:

- ◆ **Logical deletes** Do not physically delete the row in the consolidated database. Instead, have a status column that keeps track of whether rows are valid. This simplifies the *download_delete_cursor*. However, the *download_cursor* and other applications may need to be modified to recognize and use the status column. If you have a last modified column that holds the time of deletion, and if you also keep track of the last

download time for each remote, then you can physically delete the row once all remote download times are newer than the time of deletion.

- ◆ **Shadow table** For each table for which you want to track deletes, create a shadow table with two columns, one holding the primary key for the table, and the other holding a timestamp. Create a trigger that inserts the primary key and timestamp into the shadow table whenever a row is deleted. Your `download_delete_cursor` can then select from this shadow table. As with logical deletes, you can delete the row from the shadow table once all remote databases have downloaded it.

The MobiLink synchronization server deletes rows in the remote database by selecting primary key values from the consolidated database and passing those values to the remote database. If the values match those of a primary key in the remote database, then that row is deleted.

- ◆ Each `download_delete_cursor` script must contain a `SELECT` statement or a call to a stored procedure that returns a result set. The MobiLink synchronization server uses this statement to define a cursor in the consolidated database.
- ◆ This statement must select all the columns that correspond to the primary key columns in the table in the remote database. The columns in the consolidated database can have different names than the corresponding columns in the remote database, but they must be of compatible types.
- ◆ The values must be selected in the same order as the corresponding columns are defined in the remote database. That order is the order of the columns in the `CREATE TABLE` statement used to make the table, not the order they appear in the statement that defines the primary key.

While each `download_delete_cursor` script must select all the column values present in the primary key of the corresponding remote table, it may also select all the other columns. This feature is present only for compatibility with older clients. Selecting the additional columns is less efficient, as the database engine must retrieve more data. Unless the client is of an old design, the MobiLink synchronization server discards the extra values immediately. The extra values are downloaded only to older clients.

Deleting all the rows in a table

When MobiLink detects a `download_delete_cursor` with a row that contains all NULLs, it deletes all the data in the remote table. The number of NULLs in the `download_delete_cursor` can be the number of primary key columns or the total number of columns in the table.

For example, the following `download_delete_cursor` SQL script deletes every row in a table in which there are two primary key columns. This

example works for Adaptive Server Anywhere, Adaptive Server Enterprise, and Microsoft SQL Server databases.

```
SELECT NULL, NULL
```

In IBM DB2 and Oracle consolidated databases, you must specify a dummy table to select NULL. For IBM DB2 7.1, you can use the following syntax:

```
SELECT NULL FROM SYSIBM.SYSDUMMY1
```

For Oracle consolidated databases, you can use the following syntax:

```
SELECT NULL FROM DUAL
```

Examples

The following example is a `download_delete_cursor` script for a remote table that holds employee information. The MobiLink synchronization server uses this SQL statement to define the delete cursor. This script deletes information about all employees who are both in the consolidated and remote databases at the time the script is executed.

```
SELECT emp_id  
FROM employee
```

The `download_delete_cursor` accepts the parameters `last_download` and `ml_username`. The following script shows how you can use each parameter to narrow your selection.

```
SELECT order_id  
FROM ULOrder  
WHERE last_modified > ?  
      AND status = 'Approved'  
      AND user_name = ?
```

Tips

The above examples could prove inefficient in an organization with many employees. You can make the delete process more efficient by selecting only rows that could be present in the remote database. For example, you could limit the number of rows by selecting only those people who have recently been assigned a new manager. Another strategy is to allow the client application to delete the rows itself. This method is possible only when a rule identifies the unneeded rows. For example, rows might contain a timestamp that indicates an expiry date. Before you delete the rows, use the `STOP SYNCHRONIZATION DELETE` statement to stop these deletes being uploaded during the next synchronization. Be sure to execute `START SYNCHRONIZATION DELETE` immediately afterwards if you want other deletes to be synchronized in the normal fashion.

☞ You can use the referential integrity checking built into all MobiLink

clients to delete rows in a particularly efficient manner. For details, see [“Referential integrity and synchronization” on page 28](#).

☞ For more information about `download_delete_cursor`, see “`download_delete_cursor` cursor event” [*MobiLink Synchronization Reference*, page 136].

Writing scripts to handle errors

An error in a synchronization script occurs when an operation in the script fails while the MobiLink synchronization server is executing it. The DBMS returns a `SQLCODE` to the MobiLink synchronization server indicating the nature of the error. Each consolidated database DBMS has its own set of `SQLCODE` values.

When an error occurs, the MobiLink synchronization server invokes the `handle_error` event. You should provide a connection script associated with this event to handle errors. The MobiLink synchronization server passes several parameters to this script to provide information about the nature and context of the error, and requires an output value to tell it how to respond to the error.

Error handling actions

Some actions you may wish to take in an error-handling script are:

- ◆ Log the error in a separate table.
- ◆ Instruct the MobiLink synchronization server whether to ignore the error and continue, or rollback the synchronization, or rollback the synchronization and shut down the MobiLink synchronization server.
- ◆ Send an e-mail message.

☞ For more information, see “`handle_error` connection event” [*MobiLink Synchronization Reference*, page 174].

Reporting errors

Since errors can disrupt the natural progression of the synchronization process, it can be difficult to create a log of errors and their resolutions. The `report_error` script provides a means of accomplishing this task. The MobiLink synchronization server executes this script whenever an error occurs. If the `handle_error` script is defined, it is executed immediately prior to the reporting script.

The parameters to the `report_error` script are identical to those of the `handle_error` script, except that the `report_error` script can not modify the action code. Since the value of the action code is that returned by the `handle_error` script, this script can be used to debug error-handling problems.

This script typically consists of an insert statement, which records the values, perhaps with other data, such as the time or date, in a table for later reference. To ensure that this data is not lost, the MobiLink synchronization server always runs this script in a separate transaction and automatically commits the changes as soon as this script completes.

☞ For more information, see “report_error connection event” [*MobiLink Synchronization Reference*, page 194].

Example

The following report_error script, which consists of a single insert statement, adds the script parameters into a table, along with the current date and time. The script does not commit this change because the MobiLink synchronization server always does so automatically.

```
INSERT INTO errors
VALUES( CURRENT DATE, ?, ? ,?, ?, ? );
```

Handling multiple errors on a single SQL statement

ODBC allows multiple errors per SQL statement, and some DBMSs make use of this feature. Microsoft SQL Server, for example, can have two errors for a single statement. The first is the actual error, and the second is usually an informational message telling you why the current statement has been terminated.

When a single SQL statement causes multiple errors, the handle_error script is invoked once per error. The MobiLink synchronization server uses the most severe action code (that is, the numerically greatest) to determine the action to take. The same applies to the handle_error script.

If the handle_error script itself causes a SQL error, then the default action code (3000) is assumed.

Testing script syntax

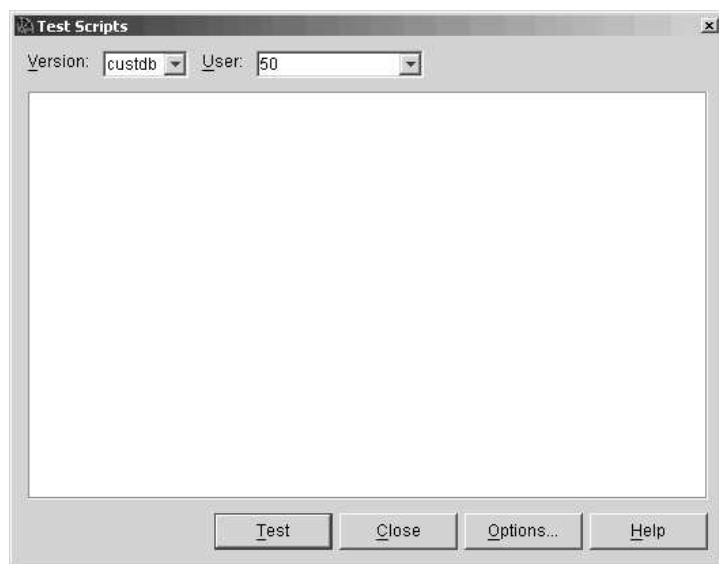
As you develop your synchronization scripts, you can use Sybase Central to test for syntax errors in your scripts.

Testing the scripts is done without any remote site in place. No data is added to the database or downloaded from the database during testing. The validity of the synchronized data itself is not tested.

❖ To test your synchronization scripts

1. Start Sybase Central and connect to a database from MobiLink Synchronization.
2. In the left pane, click the database name.
3. Right-click the Synchronized Tables folder or the Connection scripts folder and select Test Scripts from the popup menu.

The Test Scripts window appears:



4. Click Test. If prompted, enter parameters. The results of the test are displayed in the window.

The test results include a listing of which scripts are executed, in which order. They also include a listing of any syntax errors or data type errors found during the test.

DBMS-dependent scripts

Some aspects of scripts depend on the DBMS you are using. A number of factors determine the kind of scripting needed for your synchronization with your ODBC compliant database, and these factors include, but are not limited to:

- ◆ Session-wide connection variables
- ◆ User defined procedures
- ◆ Autoincrement methods

The chart below outlines the most common supported databases and their properties.

Feature	Oracle	DB2	Adaptive Server Anywhere and other
Session-wide variables	No	No	Yes
User-defined procedures	Yes	No	Yes
Autoincrement for primary keys.	No	Yes	Yes

One strategy for using MobiLink with these supported databases is to write your table scripts and synchronization logic in the DBMS version of the SQL language. Another strategy for using MobiLink with any supported consolidated database uses Java synchronization logic. When you use Java synchronization logic you can hold session-wide variables and create user-defined procedures in Java.

☞ For information about Java synchronization logic, see [“Writing Java synchronization logic” on page 232](#).

☞ For information about .NET synchronization logic, see [“Writing Synchronization Scripts in .NET” on page 251](#).

Supported DBMS scripting strategies

Following are a number of issues that you may encounter when using a consolidated database that is not Adaptive Server Anywhere.

Oracle issues

Oracle does not provide session-wide variables. You can store session-wide information in variables within Oracle packages. Oracle packages allow variables to be created, modified and destroyed and these variables may last as long as the Oracle package is current.

Oracle does not have autoincrementing primary key values. You can use an

	<p>Oracle sequence to maintain primary key uniqueness. The CustDB sample database provides coding examples, which can be found in <i>Samples\MobiLink\CustDB\custora.sql</i>.</p> <p>☞ For an example of using an Oracle sequence, see “Tutorial: Using MobiLink with an Oracle 8i Consolidated Database” on page 401.</p>
IBM DB2 issues	<p>IBM DB2 does not have the ability to make session-wide variables. It also does not support packages which would allow you to run user-defined procedures. A convenient solution is to use a base table with an extra varchar column for the MobiLink user name. This column effectively partitions the rows of the base table between concurrent synchronizations.</p> <p>IBM DB2 does not have the ability to make user-defined procedures. However, you can use Java or .NET to manipulate SQL statements and substitute new values.</p> <p>☞ For an example of Java as a procedural language for DB2, see the CustDB scripts in the files <i>Samples\MobiLink\CustDB\custdbq.sql</i> and <i>Samples\MobiLink\CustDB\custdbq.java</i>.</p> <p>☞ For more information about Java and .NET, see</p> <ul style="list-style-type: none"> ◆ “Options for writing synchronization logic” on page 31 ◆ “Writing Synchronization Scripts in Java” on page 227 ◆ “Writing Synchronization Scripts in .NET” on page 251
Adaptive Server Enterprise issues	<p>To download BLOB data from an Adaptive Server Enterprise consolidated database, you need to set an ODBC driver connection option to allow column sizes greater than 4096 bytes. To do this, use the ODBC driver connection option called <code>StaticCursorLongColBuffLen</code>. For example,</p> <pre>dbmlsrv9 -c "...;StaticCursorLongColBuffLen=number"</pre> <p>where <i>number</i> is in bytes, and is larger than the largest expected BLOB.</p> <p>Note that using this option consumes significantly more disk space on the computer that runs the MobiLink synchronization server.</p>
Storing the MobiLink user name	<p>Some database-management systems provide no convenient mechanism to store the identity of the current user.</p> <p>☞ For more information, see “Storing the user name” on page 93.</p>
Invoking procedures from scripts	<p>Some databases, such as Microsoft SQL Server, require that procedure calls with parameters be written using the ODBC syntax.</p> <pre>{ CALL procedure_name(?, ?, ...) }</pre>

On these systems, an error-handler that uses a RETURN value can also be in the following form. For example, you can return values in OUTPUT parameters in IBM DB2.

```
{ ? = CALL procedure_name( ?, ?, ... ) }
```

Adaptive Server Enterprise also requires the latter format when returning a value from a procedure.

Numeric and decimal columns

The MobiLink synchronization server requires that primary key values of type numeric or decimal be explicitly converted to their types under Adaptive Server Enterprise.

You must add an explicit conversion to the numeric parameters in the script as displayed in the following examples.

```
SELECT ...
WHERE numeric_col = CONVERT( NUMERIC, ? )
...
```

The above statement explicitly converts the first parameter to type NUMERIC.

```
SELECT ...
WHERE decimal_col = CONVERT( DECIMAL(10,8), ? )
...
```

The above statement explicitly converts the first parameter to type DECIMAL (10,8).

CHAR columns

In Adaptive Server Anywhere databases (including UltraLite), CHAR is the same as VARCHAR: values are not blank-padded to a fixed width. In many other DBMSs, CHAR data types are blank-padded to the full length of the string. The dbmlsrv9 command line option -b can be used to remove trailing blanks from strings during synchronization.

☞ For more information, see “-b option” [*MobiLink Synchronization Reference*, page 8].

Data conversion

For information about the conversion of data that must take place when a MobiLink synchronization server communicates with a consolidated database that was not made with Adaptive Server Anywhere, see “DataType Conversions” [*MobiLink Synchronization Reference*, page 323].

CHAPTER 4

Synchronization Techniques

About this chapter

This chapter describes a variety of techniques that you can use to tackle common synchronization tasks encountered in MobiLink installations.

☞ There are sample applications that provide examples of the techniques that are described in this chapter. For more information, see [“The Contact Sample Application” on page 413](#), and [“The CustDB Sample Application” on page 429](#).

The techniques in this chapter are illustrated using SQL scripts. Many of the same techniques can be implemented in Java or .NET synchronization logic. For more information, see

- ◆ [“Writing Synchronization Scripts in Java” on page 227](#)
- ◆ [“Writing Synchronization Scripts in .NET” on page 251](#)

Contents

Topic:	page
Introduction	70
Development tips	71
Timestamp-based synchronization	72
Snapshot synchronization	74
Partitioning rows among remote databases	77
Maintaining unique primary keys	81
Handling conflicts	90
Data entry	94
Handling deletes	95
Handling failed downloads	96
Downloading a result set from a stored procedure call	97
Schema changes in remote databases	100

Introduction

The chapter [“Writing Synchronization Scripts” on page 37](#) describes how to write simple synchronization scripts, store them in your database, and test that they are free of syntax errors.

Many useful synchronization features require not just one script, but a set of scripts working together. This chapter describes how to implement some common synchronization techniques. The examples describe SQL synchronization scripts. You can also use Java or .NET synchronization logic, although the upload and download events still require a knowledge of the SQL scripts.

Example

The timestamp-based synchronization of the Customer table used in the Contact sample application requires the following scripts:

- ◆ An **upload_insert** script to handle new rows added at remote databases at the consolidated database.
- ◆ An **upload_update** script to handle modifications made at remote databases at the consolidated database.
- ◆ An **upload_delete** script to handle rows deleted from remote databases at the consolidated database.
- ◆ A **download_cursor** script to download new and updated rows to remote databases.
- ◆ A **download_delete_cursor** script to download rows to be deleted from remote databases.

Development tips

Adding synchronization functionality to an application adds an added level of complexity to your application. The following tips may be useful.

- ◆ **Wait** If you try to add synchronization to a prototype application, it can be difficult to see which components are causing problems. This is particularly the case with UltraLite applications, where database and application are compiled together. When developing a prototype, temporarily hard code INSERT statements in your application to provide data for testing and demonstration purposes. Once your prototype is working correctly, enable synchronization and discard the temporary INSERT statements.
- ◆ **Go step-by-step** Start with straightforward synchronization techniques. Operations such as a simple upload or download require only one or two scripts. Once those are working correctly, you can introduce more advanced techniques, such as timestamps, primary key pools, and conflict resolution.

Following are some fundamental rules of MobiLink synchronization applications.

- ◆ Every table that is to be synchronized must have a primary key.
- ◆ Don't update the values of primary keys.
- ◆ Primary keys must be unique across all synchronized databases.
- ◆ The MobiLink user ID that identifies each remote database must be unique.

Timestamp-based synchronization

The timestamp method is the most useful general technique for efficient synchronization. The technique involves tracking the last time that each user synchronized, and using this information to control the rows downloaded to each remote database.

MobiLink maintains a timestamp value indicating when each MobiLink user last downloaded data. This value is called the **last download timestamp**. The last download timestamp is provided as a parameter to many events, and can be used in synchronization scripts.

❖ **To implement timestamp-based synchronization for a table**

- 1. At the consolidated database, add a column that holds the most recent time the row was modified. This column is not needed at remote databases. The column is typically declared as follows:

DBMS	last modified column
Adaptive Server Anywhere	timestamp DEFAULT timestamp
Adaptive Server Enterprise	datetime
Microsoft SQL Server	datetime
Oracle	date
IBM DB2	timestamp

- 2. In scripts for the download_cursor and download_delete_cursor events, compare the first parameter to the value in the timestamp column.

Example

The following table declaration and scripts implement timestamp-based synchronization on the Customer table in the Contact sample:

◆ Table definition:

```
CREATE TABLE "DBA"."Customer"(  
  "cust_id" integer NOT NULL  
    DEFAULT GLOBAL AUTOINCREMENT,  
  "name" char(40) NOT NULL,  
  "rep_id" integer NOT NULL,  
  "last_modified" timestamp NULL DEFAULT timestamp,  
  "active" bit NOT NULL,  
  PRIMARY KEY ("cust_id") )
```

◆ download_delete_cursor script:


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

◆ download_cursor script:

```
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
```

☞ For more information, see [“Synchronization logic source code”](#) on page 444, and [“Synchronizing contacts in the Contact sample”](#) on page 423.

Snapshot synchronization

Timestamp-based synchronization is appropriate for most synchronizations. However, occasionally you may want to update a snapshot of your data.

Snapshot synchronization of a table is a complete download of all relevant rows in the table, even if they have been downloaded before. This is the simplest synchronization method, but can involve unnecessarily large data sets being exchanged, which can limit performance.

You can use snapshot synchronization for downloading all the rows of the table, or in conjunction with a partitioning of the rows as described in [“Partitioning rows among remote databases” on page 77](#).

When to use snapshot synchronization

The snapshot method is typically most useful for tables that have both the following characteristics.

- ◆ **Relatively few rows** When there are few rows, the overhead associated with downloading all of them is small.
- ◆ **Rows that change frequently** When most rows in a table change frequently, there is little to be gained by explicitly excluding those that have not changed since the last synchronization.

A table that holds a list of exchange rates could be suited to this approach because there are relatively few currencies, but the rates of most change frequently. Depending on the nature of the business, a table that holds prices, a list of interest rates, or current news items could all be candidates.

❖ To implement snapshot-based synchronization

1. Leave the upload scripts undefined unless remote users update the values.
2. If the table may have rows deleted, write a `download_delete_cursor` script that deletes all the rows from the remote table, or at least all rows no longer required. Do not delete the rows from the consolidated database; rather, mark them for deletion. You must know the row values to delete them from the remote database.

☞ For more information, see [“Writing download_delete_cursor scripts” on page 58](#).

3. Write a `download_cursor` script that selects all the rows you want to include in the remote table.

Mark rows for deletion

Rather than deleting rows from the consolidated database, mark them for deletion. You must know the row values to delete them from the remote database. Select only unmarked rows in the `download_cursor` script and only marked rows in the `download_delete_cursor` script.

The `download_delete_cursor` script is executed before the `download_cursor` script. If a row is to be included in the download stream, you need not include a row with the same primary key in the delete list. When a downloaded row is received at the remote location, it replaces a preexisting row with the same primary key.

☞ For more information, see [“Writing scripts to download rows” on page 56](#).

An alternative deletion technique

Rather than delete rows from the remote database using a `download_cursor` script, you can allow the remote application to delete the rows. For example, immediately following synchronization, you could allow the application to execute SQL statements that delete the unneeded rows.

Rows deleted by the application are ordinarily uploaded to the MobiLink synchronization server upon the next synchronization, but you can prevent this upload using the `STOP SYNCHRONIZATION DELETE` statement, as follows.

```
STOP SYNCHRONIZATION DELETE;
DELETE FROM table-name
WHERE expiry_date < CURRENT_TIMESTAMP;
COMMIT;
START SYNCHRONIZATION DELETE;
```

Naturally, a different condition may be required in the `WHERE` clause, depending on the business logic of the application.

Example

The `ULProduct` table in the sample application is maintained by snapshot synchronization. The table contains relatively few rows, and for this reason, there is little overhead in using snapshot synchronization.

1. There is no upload script. This reflects a business decision that products cannot be added at remote databases.
2. There is no `download_delete_cursor`, reflecting an assumption that products are not removed from the list.
3. The `download_cursor` script selects the product identifier, price, and name of every current product. If the product is pre-existing, the price in the remote table will be updated. If the product is new, a row will be inserted in the remote table.

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

☞ For another example of snapshot synchronization in a table with very few rows, see [“Synchronizing sales representatives in the Contact sample”](#) on page 421.

Partitioning rows among remote databases

Each user of a MobiLink remote database can contain a different subset of the data in the consolidated database. Stated another way, you can write your scripts so that data is **partitioned** among remote databases.

The partitioning may be disjoint, or it may contain overlaps. For example, if each employee has their own set of customers, with no shared customers, the partitioning would be **disjoint**. If there are shared customers, who appear in more than one remote database, the partitioning contains **overlaps**.

Partitioning is implemented in the `download_cursor` and `download_delete_cursor` scripts for the table, which define the rows to be downloaded to the remote database. Each of these scripts has a single parameter, which is the synchronization user name. By defining your scripts using this parameter in the `WHERE` clause, each user gets the appropriate rows.

Disjoint partitioning

Partitioning is controlled by the `download_cursor` and `download_delete_cursor` scripts for each table involved in synchronization. These scripts take a single parameter, which is the user name supplied in the call to synchronize.

❖ To partition a table among remote databases

1. Include in the table definition a column containing the synchronization user name in the consolidated database. You need not download this column to remote databases.
2. Include a condition in the `WHERE` clause of the `download_cursor` and `download_delete_cursor` scripts requiring this column to match the script parameter.

The script parameter is represented by a question mark in the script. The user name is the second parameter in the `download_cursor` script. For example, the following `download_cursor` script partitions a table named `Contact` by employee ID.

```
SELECT id, contact_name
FROM Contact
WHERE last_modified > ?
AND emp_id = ?
```

☞ For more information, see “`download_cursor` cursor event” [MobiLink Synchronization Reference, page 133], and “`download_delete_cursor` cursor event” [MobiLink Synchronization Reference, page 136].

Example

The primary key pool tables in the CustDB sample application are used to supply each remote database with its own set of primary key values. This technique is used to avoid duplicate primary keys, and is discussed in [“Maintaining unique primary keys” on page 81](#).

A necessary feature of the method is that primary key-pool tables must be partitioned among remote databases in a disjoint fashion.

One key-pool table is ULCustomerIDPool, which holds primary key values for each user to use when they add customers. The table has three columns:

- ◆ **pool_cust_id** A primary key value for use in the ULCustomer table. This is the only column downloaded to the remote database.
- ◆ **pool_emp_id** The employee who owns this primary key.
- ◆ **last_modified** This table is maintained using the timestamp technique, based on the last_modified column.

☞ For information on timestamp synchronization, see [“Timestamp-based synchronization” on page 72](#).

The download_cursor script for this table is as follows.

```
SELECT pool_cust_id
FROM ULCustomerIDPool
WHERE last_modified > ?
      AND pool_emp_id = ?
```

When not using a variable, you should use a join or sub-selection that includes the ? placeholder.

☞ For more information, see [“Synchronizing customers in the Contact sample” on page 421](#), and [“Synchronizing contacts in the Contact sample” on page 423](#).

Partitioning with overlaps

Some tables in your consolidated database may have rows that belong to many remote databases. Each remote database has a subset of the rows in the consolidated database and the subset overlaps with other remote databases. This is frequently the case with a customer table. In this case, there is a many-to-many relationship between the table and the remote databases and there will usually be a table to represent the relationship. The scripts for the download_cursor and download_delete_cursor events need to join the table being downloaded to the relationship table.

Example

The CustDB sample application uses this technique for the ULOrder table. The ULEmpCust table holds the many-to-many relationship information between ULCustomer and ULEmployee.

Each remote database receives only those rows from the ULOrder table for which the value of the emp_id column matches the MobiLink user name.

The Adaptive Server Anywhere version of the download_cursor script for ULOrder in the CustDB application 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 , ULEmpCust ec
WHERE o.cust_id = ec.cust_id
      AND ec.emp_id = ?
      AND ( o.last_modified > ?
            OR ec.last_modified > ? )
      AND ( o.status IS NULL
            OR o.status != 'Approved' )
      AND ( ec.action IS NULL )
```

This script is fairly complex. It illustrates that the query defining a table in the remote database can include more than one table in the consolidated database. The script downloads all rows in ULOrder for which:

- ◆ the cust_id column in ULOrder matches the cust_id column in ULEmpCust,
- ◆ the emp_id column in ULEmpCust matches the synchronization user name,
- ◆ the last modification of either the order or the employee-customer relationship was later than the most recent synchronization time for this user, and
- ◆ the status is anything other than **Approved**.

The action column on ULEmpCust is used to mark columns for delete. Its purpose is not relevant to the current topic.

The download_delete_cursor script is as follows.

```
SELECT o.order_id
FROM ULOrder o, ULEmpCust ec
WHERE o.cust_id = ec.cust_id
      AND ec.emp_id = ?
      AND ( o.last_modified > ? OR
            c.last_modified > ? )
      AND ( o.status IS NULL OR
            o.status != 'Approved' )
      AND ( ec.action IS NULL )
```

This script deletes all approved rows from the remote database.

Partitioning child tables

The example above ([“Partitioning with overlaps” on page 78](#)) illustrates how to partition tables based on a criterion in some other table.

Some tables in your remote database may have disjoint subsets or overlapping subsets, but do not contain a column that determines the subset. These are child tables that usually have a foreign key (or a series of foreign keys) referencing another table. The referenced table has a column that determines the correct subset.

In this case, the `download_cursor` script and the `download_delete_cursor` script need to join the referenced tables and have a `WHERE` clause that restricts the rows to the correct subset.

☞ For an example, see [“Synchronizing contacts in the Contact sample” on page 423](#).

Maintaining unique primary keys

Every table that is to be synchronized must have a primary key, and the primary key must be unique across all synchronized databases. The values of primary keys should not be updated.

It is often convenient to use a single column as the primary key for tables. For example, each customer should be assigned a unique identification value. If all the sales representatives work in an environment where they can maintain a direct connection to the database, assigning these numbers is easily accomplished. Whenever a new customer is inserted into the customer table, automatically add a new primary key value that is greater than the last value.

In a disconnected environment, assigning unique values for primary keys when new rows are inserted is not as easy. When a sales representative adds a new customer, she is doing so to a remote copy of the Customer table. You must prevent other sales representatives, working on other copies of the Customer table, from using the same customer identification value.

This section describes the following ways to solve the problem of how to generate unique primary keys:

- ◆ Using Universally Unique IDs (UUIDs)
- ◆ Using global autoincrement values.
- ◆ Using primary key pools.

Maintaining unique primary keys using UUIDs

You can ensure that primary keys in Adaptive Server Anywhere databases are unique by using the `newid()` function to create universally unique values for your primary key. The resulting UUIDs can be converted to a string using the `uuidtostr()` function, and converted back to binary using the `strtouuid()` function.

UUIDs are unique across all computers. However, the values are completely random and so cannot be used to determine when a value was added, or the order of values. UUID values are also considerably larger than the values required by other methods (including global autoincrement), and require more table space in both the primary and foreign key tables. Indexes on tables using UUIDs are also less efficient.

☞ For more information, see

- ◆ “The NEWID default” [*ASA SQL User’s Guide*, page 82]

-
- ◆ “NEWID function [Miscellaneous]” [ASA SQL Reference, page 159]
 - ◆ “UUIDTOSTR function [STRING]” [ASA SQL Reference, page 200]
 - ◆ “STRTOUUID function [STRING]” [ASA SQL Reference, page 192]

Example

The following CREATE TABLE statement creates a primary key that is universally unique:

```
CREATE TABLE customer (  
    cust_key BINARY(16) NOT NULL  
        DEFAULT newid( )  
    rep_key VARCHAR(5)  
    PRIMARY KEY(cust_key)
```

Maintaining unique primary keys using global autoincrement

In Adaptive Server Anywhere and UltraLite databases, you can set the default column value to be GLOBAL AUTOINCREMENT. You can use this default for any column in which you want to maintain unique values, but it is particularly useful for primary keys.

❖ To use global autoincrement columns

1. Declare the column as a global autoincrement column.

When you specify default global autoincrement, the domain of values for that column is partitioned. Each partition contains the same number of values. For example, if you set the partition size for an integer column in a database to 1000, one partition extends from 1001 to 2000, the next from 2001 to 3000, and so on.

☞ See “[Declaring default global autoincrement](#)” on page 83.

2. Set the GLOBAL_DATABASE_ID value.

Adaptive Server Anywhere supplies default values in a database only from the partition uniquely identified by that database’s number. For example, if you assigned the database in the above example the identity number 10, the default values in that database would be chosen in the range 10001–11000. Another copy of the database, assigned the identification number 11, would supply default value for the same column in the range 11001–12000.

☞ See “[Setting the GLOBAL_DATABASE_ID value](#)” on page 83.

For more information

This section describes how to use global autoincrement columns in Adaptive Server Anywhere remote databases. For information on using global autoincrement columns in UltraLite databases, see “[Declaring default global autoincrement columns](#)” [UltraLite Database User’s Guide, page 151].

☞ For information on how global autoincrement columns work in Adaptive Server Anywhere databases, see [“How default values are chosen” on page 85](#). For information on how they work in UltraLite databases, see [“Declaring default global autoincrement columns” \[UltraLite Database User’s Guide, page 151\]](#).

Declaring default global autoincrement

You can set default values in your database by selecting the column properties in Sybase Central, or by including the `DEFAULT GLOBAL AUTOINCREMENT` phrase in a `CREATE TABLE` or `ALTER TABLE` statement.

Optionally, the partition size can be specified in parentheses immediately following the `AUTOINCREMENT` keyword. The partition size may be any positive integer, although the partition size is generally chosen so that the supply of numbers within any one partition will rarely, if ever, be exhausted.

For columns of type `INT` or `UNSIGNED INT`, the default partition size is $2^{16} = 65536$; for columns of other types the default partition size is $2^{32} = 4294967296$. Since these defaults may be inappropriate, especially if our column is not of type `INT` or `BIGINT`, it is best to specify the partition size explicitly.

For example, the following statement creates a simple table with two columns: an integer that holds a customer identification number and a character string that holds the customer’s name.

```
CREATE TABLE customer (  
    id    INT          DEFAULT GLOBAL AUTOINCREMENT (5000),  
    name  VARCHAR(128) NOT NULL,  
    PRIMARY KEY (id)  
)
```

In the above example, the chosen partition size is 5000.

☞ For more information on `GLOBAL AUTOINCREMENT`, see [“CREATE TABLE statement” \[ASA SQL Reference, page 361\]](#).

Setting the GLOBAL_DATABASE_ID value

When deploying an application, you must assign a different identification number to each database. You can accomplish the task of creating and distributing the identification numbers by a variety of means. One method is to place the values in a table and download the correct row to each database based on some other unique property, such as user name.

❖ To set the global database identification number

1. You set the identification number of a database by setting the value of the public option `GLOBAL_DATABASE_ID`. The identification number must be a non-negative integer.

Example

For example, the following statement sets the database identification number to 20.

```
SET OPTION PUBLIC.GLOBAL_DATABASE_ID = 20
```

If the partition size for a particular column is 5000, default values for this database are selected from the range 100001–105000.

Setting unique database identification numbers when extracting databases

If you use the extraction utility to create your remote databases, you can write a stored procedure to automate the task. If you create a stored procedure named `sp_hook_dbxtract_begin`, it is called automatically by the extraction utility. Before the procedure is called, the extraction utility creates a temporary table named `#hook_dict`, with the following contents:

name	value
extracted_db_global_id	user ID being extracted

If you write your `sp_hook_dbxtract_begin` procedure to modify the value column of the row, that value is used as the `GLOBAL_DATABASE_ID` option of the extracted database, and marks the beginning of the range of primary key values for `GLOBAL DEFAULT AUTOINCREMENT` values.

Example

Consider extracting a database for remote user **user2** with a **user_id** of 101. If you do not define an `sp_hook_dbxtract_begin` procedure, the extracted database will have `GLOBAL_DATABASE_ID` set to **101**.

If you define a `sp_hook_dbxtract_begin` procedure, but it does not modify any rows in the `#hook_dict` then the option will still be set to **101**.

If you set up the database as follows:

```

set option "PUBLIC"."Global_database_id" = '1';
create table extract_id ( next_id integer not null ) ;
insert into extract_id values( 1 );
create procedure sp_hook_dbextract_begin
as
    declare @next_id integer
    update extract_id set next_id = next_id + 1000
    select @next_id = (next_id )
    from extract_id
    commit
    update #hook_dict
    set value = @next_id
    where name = 'extracted_db_global_id'

```

Then each extracted or re-extracted database will get a different GLOBAL_DATABASE_ID. The first starts at 1001, the next at 2001, and so on.

To assist in debugging procedure hooks, *dbextract* outputs the following when it is set to operate in verbose mode:

- ◆ the procedure hooks found
- ◆ the contents of #hook_dict before the procedure hook is called
- ◆ the contents of #hook_dict after the procedure hook is called

How default values are chosen

The public option GLOBAL_DATABASE_ID in each database must be set to a unique, non-negative integer. The range of default values for a particular database is $pn + 1$ to $p(n + 1)$, where p is the partition size and n is the value of the public option GLOBAL_DATABASE_ID. For example, if the partition size is 1000 and GLOBAL_DATABASE_ID is set to 3, then the range is from 3001 to 4000.

If GLOBAL_DATABASE_ID is set to a non-negative integer, Adaptive Server Anywhere chooses default values by applying the following rules:

- ◆ If the column contains no values in the current partition, the first default value is $pn + 1$.
- ◆ If the column contains values in the current partition, but all are less than $p(n + 1)$, the next default value will be one greater than the previous maximum value in this range.
- ◆ Default column values are not affected by values in the column outside of the current partition; that is, by numbers less than $pn + 1$ or greater than $p(n + 1)$. Such values may be present if they have been replicated from another database via MobiLink synchronization.

If the public option `GLOBAL_DATABASE_ID` is set to the default value of 2147483647, a null value is inserted into the column. Should null values not be permitted, the attempt to insert the row causes an error. This situation arises, for example, if the column is contained in the table's primary key.

Because the public option `GLOBAL_DATABASE_ID` cannot be set to negative values, the values chosen are always positive. The maximum identification number is restricted only by the column data type and the partition size.

Null default values are also generated when the supply of values within the partition has been exhausted. In this case, a new value of `GLOBAL_DATABASE_ID` should be assigned to the database to allow default values to be chosen from another partition. Attempting to insert the null value causes an error if the column does not permit nulls. To detect that the supply of unused values is low and handle this condition, create an event of type **GlobalAutoincrement**.

Should the values in a particular partition become exhausted, you can assign a new database id to that database. You can assign new database id numbers in any convenient manner. However, one possible technique is to maintain a pool of unused database id values. This pool is maintained in the same manner as a pool of primary keys.

You can set an event handler to automatically notify the database administrator (or carry out some other action) when the partition is nearly exhausted. For more information, see “Defining trigger conditions for events” [ASA Database Administration Guide, page 273].

☞ For more information, see “[Setting the GLOBAL_DATABASE_ID value](#)” on page 83, and “`GLOBAL_DATABASE_ID` option [database]” [ASA Database Administration Guide, page 595].

Maintaining unique primary keys using key pools

One efficient means of solving this problem is to assign each user of the database a pool of primary key values to assign as the need arises. For example, you can assign each sales representative 100 new identification values. Each sales representative can freely assign values to new customers from his own pool.

❖ To implement a primary key pool

1. Add a new table to the consolidated database and to each remote database to hold the new primary key pool. Apart from a column for the unique value, these tables should contain a column for a user name, to identify who has been given the right to assign the value.
2. Write a stored procedure to ensure that each user is assigned enough new identification values. Assign more new values to remote users who insert many new entries or who synchronize infrequently.
3. Write a `download_cursor` script to select the new values assigned to each user and download them to the remote database.
4. Modify the application that uses the remote database so that when a user inserts a new row, the application uses one of the values from the pool. The application must then delete that value from the pool so it is not used a second time.
5. Write an `upload_cursor` script. The MobiLink synchronization server will then delete rows from the consolidated pool of values that a user has deleted from his personal value pool in the remote database.
6. Write an `end_upload` script to call the stored procedure that maintains the pool of values. Doing so has the effect of adding more values to the user's pool to replace those deleted during upload.

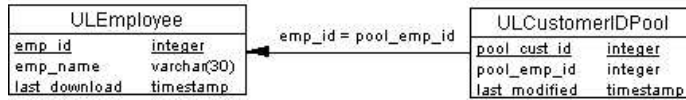
A primary key pool example

The sample application allows remote users to add customers. It is essential that each new row has a unique primary key value, and yet each remote database is disconnected when data entry is occurring.

The `ULCustomerIDPool` holds a list of primary key values that can be used by each remote database. In addition, the `ULCustomerIDPool_maintain` stored procedure tops up the pool as values are used up. The maintenance procedures are called by a table-level `end_upload` script, and the pools at each remote database are maintained by `upload_cursor` and `download_cursor` scripts.

1. The `ULCustomerIDPool` table in the consolidated database holds the pool of new customer identification numbers. It has no direct link to the `ULCustomer` table.

ULCustomer	
cust_id	integer
cust_name	varchar(30)
last_modified	timestamp



2. The `ULCustomerIDPool_maintain` procedure updates the `ULCustomerIDPool` table in the consolidated database. The following sample code is for an Adaptive Server Anywhere consolidated database.

```

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

```

This procedure counts the numbers presently assigned to the current user, and inserts new rows so that this user has a sufficient supply of customer identification numbers.

This procedure is called at the end of the upload stream, by the `end_upload` table script for the `ULCustomerIDPool` table. The script is as follows:

```
CALL ULCustomerIDPool_maintain( ? )
```

3. The `download_cursor` script for the `ULCustomerIDPool` table downloads new numbers to the remote database.

```

SELECT pool_cust_id
FROM ULCustomerIDPool
WHERE pool_emp_id = ?
AND last_modified > ?

```

4. To insert a new customer, the application using the remote database must select an unused identification number from the pool, delete this number

from the pool, and insert the new customer information using this identification number. The following embedded SQL function for an UltraLite application retrieves a new customer number from the pool.

```
bool CDemoDB::GetNextCustomerID( void )
/*****
{
    short    ind;

    EXEC SQL SELECT min( pool_cust_id )
    INTO :m_CustID:ind FROM ULCustomerIDPool;
    if( ind < 0 ) {
        return false;
    }
    EXEC SQL DELETE FROM ULCustomerIDPool
    WHERE pool_cust_id = :m_CustID;
    return true;
}
```

5. The `upload_cursor` script deletes numbers from the consolidated pool of numbers once they have been used and hence deleted from the remote pool.

```
SELECT pool_cust_id
FROM ULCustomerIDPool
WHERE pool_cust_id = ?
```

Handling conflicts

Conflicts arise during the upload of rows to the consolidated database. If two users modify the same row, a conflict is detected when the second of the rows arrives at the MobiLink synchronization server. When conflicts can occur, you should define a process to compute the correct values, or at least to log the conflict.

Conflicts are detected only during updates of a row. If an attempt to insert a row finds that the row has already been inserted, an error is generated. If an attempt to delete a row finds that the row has already been deleted, the attempt to delete is ignored. An attempt to update a row that has been deleted is a conflict.

No conflicts arise in the remote database as a result of synchronization. If a downloaded row contains a new primary key, the values are inserted into a new row. If the primary key matches that of a pre-existing row, the other values in the row are updated.

Conflicts are not the same as errors. Conflict handling can be an integral part of a well-designed application, allowing concurrency, even in the absence of locking.

Caution

Don't update primary keys in a MobiLink environment.

How conflicts are detected

Whenever a row is updated at a remote database, a copy of the values the row contained at the time of last synchronization is retained. When you next synchronize, your remote database contains not only the present data, but also a record of the values that were present the last time you synchronized.

When the client sends an updated row to the MobiLink synchronization server, it includes not only the new values, but also a copy of the original values.

Detecting conflicts

When using `upload_update` scripts, conflict detection is carried out in one of the following circumstances:

- ◆ An `upload_fetch` script is supplied.

The `upload_fetch` script typically selects a single row of data from a table corresponding to the row being updated. A typical `upload_fetch` script would conform to the following syntax:

```
SELECT col1, col2, ...
FROM table-name
WHERE pk1 = ? AND pk2 = ? ...
```

☞ For more information, see “upload_fetch table event” [*MobiLink Synchronization Reference*, page 216].

- ◆ The **upload_update** script provides a parameter for each element on the row.

The parameters for an **upload_update** event are arranged so that statements with the following syntax update rows correctly:

```
UPDATE table-name
SET col1 = ?, col2 = ?, ...
WHERE pk1 = ? AND pk2 = ? ...
```

In this statement, col1, col2 and so on are the non-primary key columns, while pk1, pk2 and so on are primary key columns.

For a conflict to be detected, the syntax must be as follows:

```
UPDATE table-name
SET col1 = ?, col2 = ?, ...
WHERE pk1 = ? AND pk2 = ? ...
AND col1 = ? AND col2 = ? ...
```

☞ For more information, see “upload_update table event” [*MobiLink Synchronization Reference*, page 231].

The MobiLink synchronization server processes each uploaded update using the following procedure.

1. MobiLink synchronization server detects conflicts only if an upload_fetch or appropriate upload_update script is applied:
 - ◆ If an upload_fetch script is supplied, the MobiLink synchronization server compares the old uploaded values to the values of the row returned by the upload_fetch statement with the same primary key values.
 - ◆ If an upload_update script of the above form is supplied, the MobiLink synchronization server compares the old uploaded values to the values of the row returned in the final set of parameters.
2. If any of the old uploaded values *do not* match the current consolidated values, the MobiLink synchronization server detects a conflict.
 - ◆ The MobiLink synchronization server inserts the old values as defined by the upload_old_row_insert script.

☞ For more information, see “upload_old_row_insert table event” [*MobiLink Synchronization Reference*, page 222].

-
- ◆ The MobiLink synchronization server inserts the new values as defined by the `upload_new_row_insert` script.
 - ☞ For more information, see “`upload_new_row_insert` table event” [*MobiLink Synchronization Reference*, page 220].
 - ◆ The MobiLink synchronization server executes the `resolve_conflict` script. In this script you can either call a stored procedure, or define a sequence of steps to resolve the conflict as appropriate.
 - ☞ For more information, see “`resolve_conflict` table event” [*MobiLink Synchronization Reference*, page 199].

You can resolve conflicts as they occur using the `resolve_conflict` script, or you can resolve all conflicts at once using the table’s `end_upload` script.

☞ For an example of conflict resolution using statement-based uploads, see “[Synchronizing products in the Contact sample](#)” on page 425.

Forced conflict resolution

Forced conflict resolution is a special technique that forces every uploaded row to be treated as if it were a conflict.

Forced conflict resolution

If no `upload_insert`, `upload_update`, or `upload_delete` script is defined for a remote table, the MobiLink synchronization server uses *forced conflict resolution*. In this mode of operation, MobiLink synchronization server attempts to insert all uploaded rows from that table using the statements defined by the `upload_old_row_insert` and `upload_new_row_insert` scripts. In essence, all uploaded rows are then treated as conflicts. You can write stored procedures or scripts to process the uploaded values in any way you want.

Without any of the `upload_insert`, `upload_update`, or `upload_delete` scripts, the normal conflict-resolution procedure is bypassed. This technique has two principal uses.

- ◆ **Arbitrary conflict detection and resolution** The automatic mechanism only detects errors when updating a row, and only then when the old values do not match the present values in the consolidated database.

You can capture the raw uploaded data using the `upload_old_row_insert` and `upload_new_row_insert` scripts, then process the rows as you see fit.
- ◆ **Performance** When the `upload_insert`, `upload_update`, or `upload_delete` are not defined, the MobiLink synchronization server is relieved of its normal conflict-detection tasks, which involve querying the consolidated database one row at a time. Instead, it needs only to insert the raw

uploaded information using the statements defined by the `upload_old_row_insert` and `upload_new_row_insert` scripts. Since only inserts are involved, the MobiLink synchronization server performs these inserts using bulk operations that are more efficient.

Storing the user name

When you write `upload_old_row_insert` or `upload_new_row_insert` scripts, you can include an extra column in your select statement. If you do so, the MobiLink synchronization server automatically inserts the user name into the first column, and then uses the rest of the columns as usual. This mechanism is available because some database-management systems provide no convenient mechanism to store the identity of the current user.

You can use this feature to conveniently identify which user inserted each row. This information allows you to include user-specific logic in the `resolve_conflict` script.

For example, an ordinary `upload_old_row_insert` script is of the following form. The items in the select list correspond to the columns of the remote table.

```
INSERT c1, c2, . . . , cN FROM table
```

However, the following syntax is also permitted.

```
INSERT user_name, c1, c2, . . . , cN FROM table
```

Normally, the selected columns must match the columns of the remote table in both number and type. This case is an exception. The single extra column in the select list must be of a type suitable to hold the user name, for example, `VARCHAR(128)`. The subsequent columns in the list must match the columns of the remote table in order and type, as usual. If you include more than one extra column, an error results.

☞ For more information, see “`upload_old_row_insert` table event” [*MobiLink Synchronization Reference*, page 222] and “`upload_new_row_insert` table event” [*MobiLink Synchronization Reference*, page 220].

Data entry

In some databases, there are tables that are only used for data entry. One way of processing these tables is to upload all inserted rows at each synchronization, and remove them from the remote database on the download stream. After synchronization, the remote table is empty again, ready for another batch of data.

To achieve this model, you can upload rows into a temporary table and then insert them into a base table using an `end_upload` table script. The temporary table can be used in the `download_delete_cursor` to remove rows from the remote database following a successful synchronization.

Alternatively, you can allow the client application to delete the rows, using the `STOP SYNCHRONIZATION DELETE` statement to stop the deletes being uploaded during the next synchronization.

☞ For more information, see “`STOP SYNCHRONIZATION DELETE` statement [MobiLink]” [*MobiLink Synchronization Reference*, page 260].

Handling deletes

When rows are deleted from the consolidated database, there needs to be a record of the row so it can be removed from any remote databases that have the row.

One technique is to not delete the row. Data that is no longer required can be marked as inactive by changing a status column in the row. The `download_cursor` and `download_delete_cursor` can refer to the status of the row in the WHERE clause. The CustDB sample application uses this technique for the ULOrder table using the status column, and the Contact sample uses the technique on the Customer, Contact, and Product tables.

This technique is used in the ULEmpCust table in the CustDB sample application, in which the action column holds a D for Delete. The scripts use this value to delete the record from the remote database, and delete the record from the consolidated database at the end of the synchronization.

A second technique is to have a shadow table that stores the primary key values of deleted rows. When a row is deleted, a trigger can populate the shadow table. The `download_delete_cursor` can use the shadow table to remove rows from remote databases. The shadow table only needs to have the primary key columns from the real table.

☞ For more information, see

- ◆ “download_cursor cursor event” [*MobiLink Synchronization Reference*, page 133]
- ◆ [“Writing download_delete_cursor scripts” on page 58](#)
- ◆ “download_delete_cursor cursor event” [*MobiLink Synchronization Reference*, page 136]
- ◆ [“Snapshot synchronization” on page 74](#)
- ◆ [“Temporarily stopping synchronization of deletes” on page 193](#)
- ◆ “STOP SYNCHRONIZATION DELETE statement [MobiLink]” [*MobiLink Synchronization Reference*, page 260]

Handling failed downloads

Bookkeeping information about what is downloaded must be maintained in the download transaction. This information is updated atomically with the download being applied to the remote database.

If a failure occurs before the entire download stream is applied to the remote database, by default the MobiLink synchronization server does not get confirmation for the download and rolls back the download transaction. Since the bookkeeping information is part of the download transaction, it is also rolled back. Next time the download stream is built, it will use the original bookkeeping information. You can change this default behavior. For more information, see “SendDownloadACK (sa) extended option” [*MobiLink Synchronization Reference*, page 62] or “Send Download Acknowledgement synchronization parameter” [*UltraLite Database User’s Guide*, page 172].

When testing your synchronization scripts, you should add logic to your end_download script that causes occasional failures. This will ensure that your scripts can handle a failed download.

Downloading a result set from a stored procedure call

You can download a result set from a stored procedure call. For example, you might currently have a `download_cursor` for the following table:

```
CREATE TABLE MyTable (
    pk INTEGER PRIMARY KEY NOT NULL,
    col1 VARCHAR(100) NOT NULL,
    col2 VARCHAR(20) NOT NULL
)
```

The `download_cursor` cursor script might look as follows:

```
SELECT pk, col1, col2
FROM MyTable
WHERE last_modified > ?
AND employee = ?
```

If you want your downloads to MyTable to use more sophisticated business logic, you can now create your script as follows, where `DownloadMyTable` is a stored procedure taking two parameters (last-download timestamp and MobiLink user name) and returning a result set. (This example uses an ODBC calling convention for portability):

```
{call DownloadMyTable( ?, ? )}
```

Following are some simple examples for each supported consolidated database. Consult the documentation for your consolidated database for full details.

The following example works with Adaptive Server Anywhere, Adaptive Server Enterprise, and Microsoft SQL Server.

```
CREATE PROCEDURE SPDownload
    @last_dl_ts DATETIME,
    @u_name VARCHAR( 128 )
AS
BEGIN
    SELECT pk, col1, col2
    FROM MyTable
    WHERE last_modified > @last_dl_ts
    AND employee = @u_name
END
```

The following example works with Oracle. Oracle requires that a package be defined. This package must contain a record type for the result set, and a cursor type that returns the record type.

Note

This example requires that Oracle return a result set. In the ODBC Oracle Driver Setup dialog, you must select the Procedure Returns Results option; or in the connection string, set ProcedureRetResults=1. For more information about setting up the Oracle ODBC driver, see “iAnywhere Solutions ODBC Driver for Oracle Wire Protocol” [*iAnywhere Solutions ODBC Drivers*, page 31].

```
Create or replace package SPInfo as
Type SPRec is record (
    pk      integer,
    col1    varchar(100),
    col2    varchar(20)
);
Type SPCursor is ref cursor return SPRec;
End SPInfo;
```

Next, Oracle requires a stored procedure with the cursor type as the first parameter. Note that the download_cursor script only passes in two parameters, not three. For stored procedures returning result sets in Oracle, cursor types declared as parameters in the stored procedure definition define the structure of the result set, but do not define a true parameter as such.

```
Create or replace procedure
    DownloadMyTable( spcursor IN OUT SPInfo.SPCursor,
                    last_dl_ts IN DATE,
                    user_name IN VARCHAR ) As
Begin
    Open spcursor For
        select pk, col1, col2
        from MyTable
            where last_modified > last_dl_ts
            and employee = user_name;
End;
```

The following example works with IBM DB2 UDB.

```
CREATE PROCEDURE DownloadMyTable(
    IN last_dl_ts TIMESTAMP,
    IN u_name VARCHAR( 128 ) )
    EXTERNAL NAME 'DLMyTable!DownloadMyTable'
    RESULT SETS 1
    FENCED
    LANGUAGE JAVA PARAMETER STYLE DB2GENERAL
```

The following example is a Java implementation of the stored procedure, in DLMyTable.java. To return a result set, you must leave the result set open when the method returns:

```
import COM.ibm.db2.app.*;
import java.sql.*;

public class DLMyTable extends StoredProc
{
    public void DownloadMyTable(
        Date last_dl_ts,
        String u_name ) throws Exception
    {
        Connection conn = getConnection();
        conn.setAutoCommit( false );
        Statement s = conn.createStatement();
        // Execute the select and leave it open.
        ResultSet r = s.executeQuery(
            "select pk, col1, col2 from MyTable"
            + " where last_modified > '"
            + last_dl_ts
            + "' and employee = '"
            + u_name + "'" );
    }
}
```

Schema changes in remote databases

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

Adaptive Server
Anywhere remote
databases

❖ To add tables to Adaptive Server Anywhere remote databases

1. Add the associated table scripts in the consolidated database.

The same script version may be used for the remote database without the new table and the remote database with the new table. However, if the presence of the new table changes how existing tables are synchronized, then you must create a new script version, and create new scripts for all tables being synchronized with the new script version.

2. Perform a normal synchronization.

This step is optional, but recommended, before changing schema.

3. Use the ALTER PUBLICATION statement to add the table. For example,

```
ALTER PUBLICATION your_pub  
ADD TABLE table_name
```

☞ For more information, see “ALTER PUBLICATION statement”
[*MobiLink Synchronization Reference*, page 234].

4. Synchronize. Use the new script version, if required.

Changing table
definitions in remote
databases

Changing the number or type of columns in an existing table must be done carefully. When a MobiLink client synchronizes with a new schema, it expects scripts, such as upload_update or download_cursor, which have parameters for all columns in the remote table. An older remote database expects scripts that have only the original columns.

❖ To alter a published table in a deployed Adaptive Server Anywhere remote database

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

☞ For more information, see “Script versions” on page 49.

2. For your new script version, create scripts for all tables in the publication(s) that contain the table that you want to alter and that are synchronized with the old script version.

3. At the remote database, perform a normal synchronization using the old script version.

4. At the remote database, use the ALTER PUBLICATION statement to temporarily drop the table from the publication. For example,

```
ALTER PUBLICATION your_pub
DROP TABLE table_name
```

☞ For more information, see “ALTER PUBLICATION statement” [*MobiLink Synchronization Reference*, page 234].

5. At the remote database, use the ALTER TABLE statement to alter the table.

☞ For more information, see “ALTER TABLE statement” [*ASA SQL Reference*, page 250].

6. At the remote database, use the ALTER PUBLICATION statement to add the table back into the publication.

☞ For more information, see “ALTER PUBLICATION statement” [*MobiLink Synchronization Reference*, page 234].

7. Synchronize with the new script version.

Note: Steps 4 through 6 may also be performed by the `sp_hook_dbmlsync_schema_upgrade` stored procedure. For more information, see “`sp_hook_dbmlsync_schema_upgrade`” [*MobiLink Synchronization Reference*, page 291].

☞ For more information about changing schemas for Adaptive Server Anywhere remote databases, see “`sp_hook_dbmlsync_schema_upgrade`” [*MobiLink Synchronization Reference*, page 291].

UltraLite remote databases

You can change the schema of a remote database by deploying a new application or through a schema upgrade.

- ◆ If you deploy a new application without a schema upgrade, you need to repopulate the UltraLite database by synchronizing with the MobiLink synchronization server.
- ◆ In the schema upgrade case, your data will be preserved. It is usually impractical to have all users upgrade to the new version of the application at the same time.

You need to be able to have both versions co-existing in the field and synchronizing with a single consolidated database. You can create two or more versions of the synchronization scripts that are stored in the consolidated database and control the actions of the MobiLink synchronization server. Each version of your application can then select the

appropriate set of synchronization scripts by specifying the correct version name when it initiates synchronization.

☞ For more information about schemas in UltraLite, see “Databases and schema files” [*UltraLite Database User’s Guide*, page 28].

CHAPTER 5

Authenticating MobiLink Users

About this chapter

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

Contents

Topic:	page
About MobiLink users	104
Choosing a user authentication mechanism	107
User authentication architecture	108
Providing initial passwords for users	110
Synchronizations from new users	111
Prompting end users to enter passwords	112
Changing passwords	113
Custom user authentication	114

About MobiLink users

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

MobiLink user names and passwords are not the same as database user names and passwords. MobiLink user names and passwords are used to identify, and optionally authenticate, clients attempting to connect to the MobiLink synchronization server.

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

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

UltraLite user
authentication

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

☞ For more information about UltraLite user authentication, see “User authentication” [*UltraLite Database User’s Guide*, page 38].

Creating MobiLink users

You can use any of the following methods to register user names in the consolidated database:

- ◆ Use the `dbmluser` utility.

☞ For more information, see “MobiLink user authentication utility” [*MobiLink Synchronization Reference*, page 308].

- ◆ Use Sybase Central.

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

☞ For more information, see “`-zu` option” [*MobiLink Synchronization Reference*, page 31].

The MobiLink user must already exist in a remote database. To add users at the remote, you have the following options:

- ◆ For Adaptive Server Anywhere remotes, use the CREATE SYNCHRONIZATION USER statement.
 - ☞ For more information, see “CREATE SYNCHRONIZATION USER statement [MobiLink]” [ASA SQL Reference, page 351].
- ◆ For UltraLite remotes, you can use the user_name field of the ul_synch_info structure. In Java, use the SetUserName() method of the ULSynchInfo class before synchronizing.
 - ☞ For more information, see “User Name synchronization parameter” [UltraLite Database User’s Guide, page 177], and “Password synchronization parameter” [UltraLite Database User’s Guide, page 167].

Sharing MobiLink user names

If you want two or more remote databases (UltraLite or Adaptive Server Anywhere) to share the same MobiLink user name, then you can create a MobiLink user name that is a base name with a unique suffix. You do this in the CREATE SYNCHRONIZATION USER statement by adding a + after the user name, followed by your suffix.

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

Example

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

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

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

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

```
CALL sp_ML_modify_user( ? )
```

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

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

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

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

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

Choosing a user authentication mechanism

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

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

- ◆ **No MobiLink user authentication** If your data is such that you do not need password protection, you can choose not to use any user authentication in your installation.
- ◆ **Built-in MobiLink user authentication** MobiLink uses the user names and passwords stored in the ml_user MobiLink system table to perform authentication.

The built-in mechanism is described in the following sections.

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

☞ For more information about custom user authentication mechanisms, see [“Custom user authentication” on page 114](#).

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

- ◆ [“Transport-Layer Security” on page 337](#)
- ◆ “Encrypting an UltraLite database” [*UltraLite Database User’s Guide*, page 36]
- ◆ “Keeping Your Data Secure” [*SQL Anywhere Studio Security Guide*, page 3]

User authentication architecture

The MobiLink user authentication system relies on user names and passwords. You can choose either to let the MobiLink synchronization server validate the user name and password using a built-in mechanism, or you can implement your own custom user authentication mechanism.

In the built-in authentication system, both the user name and the password are stored in the `ml_user` MobiLink system table in the consolidated database. The password is stored in hashed form so that applications other than the MobiLink synchronization server cannot read the `ml_user` table and reconstruct the original form of the password. You add user names and passwords to the consolidated database using Sybase Central or the `dbmluser` utility.

☞ For more information, see “MobiLink user authentication utility” [*MobiLink Synchronization Reference*, page 308].

When a MobiLink client connects to a MobiLink synchronization server, it provides the following values.

- ◆ **user name** The MobiLink user name. Mandatory. This value typically matches exactly a user name in the `ml_user` MobiLink system table.
- ◆ **password** The MobiLink password. Optional only if the user is unknown or if the corresponding password in the `ml_user` MobiLink system table is NULL.
- ◆ **new password** A new MobiLink password. Optional. MobiLink users can change their password by setting this value.

The MobiLink synchronization server, upon receiving a connection request from a MobiLink client, proceeds as follows.

If the MobiLink synchronization server finds the supplied user name in the `ml_user` MobiLink system table, compares the supplied password with the stored value. If the passwords match or the stored password is NULL, synchronization proceeds. Otherwise, the synchronization server denies the request and returns an error code to the client.

New users and passwords

If a MobiLink client supplies a user name that is not present in the `ml_user` table, the behavior is determined by a MobiLink synchronization server command line option.

For more information, see “[Synchronizations from new users](#)” on page 111.

Custom authentication

Optionally, you can substitute your own user authentication mechanism. You

do so by providing an `authenticate_user` script. If this script exists, it is executed instead of the password comparison. The script must return error codes to indicate the success or failure of the authentication.

The following sections describe how to implement the different pieces of the authentication system, and describe some specific issues you may encounter.

Providing initial passwords for users

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

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

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

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

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

❖ To provide initial MobiLink passwords (command line)

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

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

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

☞ For information about `dbmluser`, see “MobiLink user authentication utility” [*MobiLink Synchronization Reference*, page 308].


Synchronizations from new users

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

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

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

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

- ◆ **New user, no password** If the user supplied no password, then by default the user name is added to the `ml_user` table with a NULL password. This behavior provides compatibility with earlier releases of MobiLink that did not allow user authentication.
 For more information, see “`-zu` option” [*MobiLink Synchronization Reference*, page 31].
- ◆ **New user, password** If the user supplies a password, then the user name and password are both added to the `ml_user` table and the new user name becomes a recognized name in your MobiLink system.
- ◆ **New user, new password** A new user may provide information in the new password field, instead of or as well as in the password field. In either case, the new password setting overrides the password setting, and the new user is added to the MobiLink system using the new password.

Preventing
synchronization by
unknown users

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

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

Prompting end users to enter passwords

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

❖ To prompt your end users to enter their MobiLink passwords

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

- ◆ **UltraLite** When synchronizing, the UltraLite client must supply a valid value in the password field of the synchronization structure (C/C++) or object (Java). For built-in MobiLink synchronization, a valid password is one that matches the value in the ml_user MobiLink system table.

Your application should prompt the end user to enter their MobiLink user name and password before synchronizing.

☞ For more information, see “Synchronization for UltraLite Applications” [*UltraLite Database User's Guide*, page 143].

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

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

☞ For more information, see “MobiLink synchronization client” [*MobiLink Synchronization Reference*, page 36].

Changing passwords

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

❖ To prompt your end users to enter MobiLink passwords

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

◆ **UltraLite** When synchronizing, the application must supply the existing password in the password field of the synchronization structure and the new password in the new_password field.

☞ For more information, see “Password synchronization parameter” [*UltraLite Database User’s Guide*, page 167] and “New Password synchronization parameter” [*UltraLite Database User’s Guide*, page 166].

◆ **Adaptive Server Anywhere** Supply a valid existing password together with the new password on the dbmlsync command line, or in the dbmlsync connection dialog if you do not supply command line parameters.

☞ For more information, see “MobiLink synchronization client” [*MobiLink Synchronization Reference*, page 36].

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

☞ An initial password can be set in the consolidated server or on the first synchronization attempt. For more information, see [“Providing initial passwords for users” on page 110](#) and [“Synchronizations from new users” on page 111](#).

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

Custom user authentication

You can choose to use a user authentication mechanism other than the built-in MobiLink mechanism. Reasons for using a custom user authentication mechanism include integration with existing DBMS user authentication schemes, or supplying custom features, such as minimum password length or password expiry, that do not exist in the built-in MobiLink mechanism.

There are three custom authentication tools:

- ◆ dbmlsrv9 -zu option
- ◆ authenticate_user script
- ◆ authenticate_parameters script

The dbmlsrv9 -zu option allows you to control the automatic addition of users. For example, specify -zu+ to have all unrecognized MobiLink user names added to the ml_user table when they first synchronize. The -zu option works with built-in MobiLink authorization.

The authenticate_user script and authenticate_parameters script both override the default MobiLink user authentication mechanism. Use authenticate_user to create custom authentication of user IDs and passwords. Use authenticate_parameters to create custom authentication that depends on values other than user IDs and passwords.

For more information, see

- ◆ “-zu option” [*MobiLink Synchronization Reference*, page 31]
- ◆ “authenticate_user connection event” [*MobiLink Synchronization Reference*, page 100]
- ◆ “authenticate_parameters connection event” [*MobiLink Synchronization Reference*, page 98]

Java and .NET user authentication

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

A simple sample is included in the directory *Samples\MobiLink\JavaAuthentication*. The sample code in *Samples\MobiLink\JavaAuthentication\CustEmpScripts.java* implements a simple user authentication system. On the first synchronization, a MobiLink user name is added to the login_added table. On subsequent

synchronizations, a row is added to the login_audit table. In this sample, there is no test before adding a user ID to the login_added table.

For a .NET sample that explains user authentication, see [“.NET synchronization example” on page 266](#).

SQL user authentication A typical authenticate_user SQL script would be a call to a stored procedure that uses the parameters. The order of the parameters in the call must match the order above. For example, in an Adaptive Server Anywhere consolidated database, the format would be as follows:

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

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

A Transact-SQL format would be as follows:

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

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

CHAPTER 6

File-Based Downloads

About this chapter

This chapter describes an alternative way to download data to Adaptive Server Anywhere remote databases. Downloads can be distributed as files, enabling offline distribution of synchronization changes. This allows you to create a file once and distribute it to many remote databases.

Contents

Topic:	page
Introduction	118
Setting up file-based downloads	119
Validation checks	123
Examples	127

Introduction

With file-based downloads, you can put download synchronization changes in a file and transfer it to Adaptive Server Anywhere remote databases in any way a file can be transferred. For example,

- ◆ broadcast the data by satellite multicast
- ◆ apply the update using Sybase Manage Anywhere
- ◆ e-mail or ftp the file to users

You choose the users you want to receive the file. Full synchronization integrity is preserved in file-based downloads, including conflict detection and resolution. You can ensure that the file is secure by applying third-party encryption on the file.

When to use

File-based downloads are useful when a large amount of data changes on the consolidated database, but the remote database does not update the data frequently or does not do any updates at all. For example, price lists, product lists, and code tables.

File-based downloads are not useful when the downloaded data is updated frequently on the remote database or when you are running frequent upload-only synchronizations. In these situations, the remote sites may be unable to apply download files because of integrity checks that are performed when download files are applied.

Notes

- ◆ File-based downloads cannot be used as the sole means of updating remote databases. You still need to regularly perform full synchronizations or upload-only synchronizations. Full or upload-only synchronizations are required to advance log offsets and to maintain the log file, which otherwise will grow large and slow down synchronization. A full synchronization may also be required to recover from errors.
- ◆ File-based downloads currently can be used only with Adaptive Server Anywhere remote databases.

Setting up file-based downloads

To set up file-based download, you:

- ◆ Create a file-definition database.
- ◆ At the consolidated database, create scripts with a new script version.
- ◆ Create a download file.
- ◆ Apply the download file.

Create the file-definition database

To set up file-based downloads, you create a **file-definition database**. This is an Adaptive Server Anywhere database that has the same synchronization tables and publications as your remote databases. It can be located anywhere. This database contains no data or state information. It does not have to be backed up or maintained; in fact, you can delete it and recreate it as needed.

The file-definition database must include the following:

- ◆ the same publications as the remote databases, as well as the tables and columns used in the publication, the foreign key relationships and constraints of those tables and columns, and the tables required by those foreign key relationships.
- ◆ a MobiLink user name that identifies the group of remote databases that are to apply the download file. You will use this group MobiLink user name in your synchronization scripts to identify the group of remote databases.

Changes at the consolidated database

On the consolidated database, create a new script version for your file-based downloads, and implement any scripts required by your existing synchronization system into it. (Upload scripts are not required.) This script version will be used only for file-based downloads. For this script version, all scripts that take MobiLink user names as parameters will instead take a MobiLink user name that refers to a group of remote databases. This is the user name that is defined in the file-definition database.

For each script version that you have defined, implement a `begin_publication` script.

For timestamp-based downloads, implement a `modify_last_download_timestamp` script for each script version. How you

implement this script depends on how much data you intend to send in each download file. For example, one approach is to use the earliest time that any user from the group last downloaded successfully. Remember that the `ml_username` parameter passed to this script is actually the group name.

☞ For more information, see “`modify_last_download_timestamp` connection event” [*MobiLink Synchronization Reference*, page 180].

Creating the download file

The download file contains the data to be synchronized. To create the download file, set up your file-definition database and consolidated database as described above. Run `dbmlsync` with the `-bc` option and supply a file name with the extension `.df`. For example,

```
dbmlsync -c "uid=dba;pwd=sql;eng=fbd1_eng;dbf=fdef.db" -v+  
-e "sv=filebased" -bc file1.df
```

Optionally, you can specify options when you create the download file:

- ◆ **-be option** Use `-be` to add a string to the download file that can be accessed at the remote database using the `sp_hook_dbmlsync_validate_download_file` stored procedure.

☞ For more information, see “`-be option`” [*MobiLink Synchronization Reference*, page 41] and “`sp_hook_dbmlsync_validate_download_file`” [*MobiLink Synchronization Reference*, page 295].

- ◆ **-bg option** Use the `-bg` option to create a download file that can be used by remotes that have never synchronized.

Use the `-bg` option to create a download file that can be used by remotes that have never synchronized.

Synchronizing new remotes

If you want to apply a download file to a remote database that has never synchronized using MobiLink, then before you apply the download file you need to either perform a normal synchronization on the remote database or use the `dbmlsync -bg` option when creating the download file.

For timestamp-based synchronization, doing either of these two things causes the download of an initial snapshot of the data. For both timestamp and snapshot based synchronization, this step sets the generation number to the value that is generated by the `begin_publication` script on the consolidated database.

Perform a normal
synchronization

You can prepare a remote database to receive download files by performing a synchronization that does not use a download file.

Use the -bg option

Alternatively, you can create a download file with the -bg option to use with remotes that haven't yet synchronized. You apply this initial download file to prepare the remote database for file-based synchronization.

- ◆ **Snapshot downloads** If you are performing snapshot downloads, then the initial download file just needs to set the generation number. You may choose to include an initial snapshot of the data in this file, but since each snapshot download contains all the data and does not depend on previous downloads, this is not required.

For snapshot downloads, using the -bg option is straightforward. Just specify -bg in the dbmlsync command line when you create the download file. You can use the same script version to create the initial download file as you use for subsequent download files.

- ◆ **Timestamp-based downloads** If you are performing timestamp-based downloads, then the initial download must set the generation number on the remote database and include a snapshot of the data. With timestamp-based downloads, each download builds on previous ones. Each download file contains a last download timestamp. All rows changed on the consolidated after the file's last download timestamp are included in the file. To apply a file, a remote database must already have received all the changes that occurred before the file's last download timestamp. This is confirmed by checking that the file's last download timestamp is greater than or equal to the remote database's last download timestamp (the time up to which the remote database has received all changes from the consolidated database).

Before a remote can apply its first normal download file, it must receive all data changed before that file's last download timestamp and after January 1, 1900. The initial download file created with the -bg option must contain this data. The easiest way to select this data is to create a separate script version that uses the same download_cursors as your normal file-based synchronization script version but does not have a modify_last_download_timestamp script. If no modify_last_download_timestamp script is defined, then the last download timestamp for a file-based download will default to January 1, 1900.

If you apply download files built with the -bg option to remote databases that have already synchronized, the -bg option causes the generation numbers on the remote database to be updated with the value on the consolidated database at the time the download file was created. This defeats the purpose of generation numbers, which is to prevent you from applying further file-based downloads until an upload has been performed in situations such as when recovering a consolidated database that is lost or corrupted.

☞ For more information about generation numbers, see [“MobiLink generation numbers” on page 125](#).

Validation checks

Before applying a download file to a remote database, dbmlsync does a number of things to ensure that the synchronization is valid.

- ◆ dbmlsync checks the download file to ensure that the file-definition database that was used to create it has:
 - the same publication as the remote database
 - the same tables and columns used in the publication
 - the same foreign key relationships and constraints as those tables and columns
- ◆ dbmlsync checks to see if there is any data in the publication that has not been uploaded from the remote. If there is, the download file is not applied, because applying the download file could cause pending upload data to be lost.
- ◆ dbmlsync checks the last download timestamp, next last download timestamp, and creation time of the download file to ensure that:
 - newer data on the remote database will not be overwritten by older data contained in the download file.
 - a download file will not be applied if applying it means that the remote database would miss some changes that have occurred on the consolidated database. This situation might occur if the remote did not apply previous file-based downloads.

☞ For more information, see [“Automatic validation” on page 123](#).
- ◆ Optionally, dbmlsync checks the generation number in the remote database to ensure it matches the generation number in the download file.

☞ For more information, see [“MobiLink generation numbers” on page 125](#).
- ◆ Optionally, you can create custom validation logic with the `sp_hook_dbmlsync_validate_download_file` stored procedure.

☞ For more information, see [“Custom validation” on page 126](#).

Automatic validation

Before applying a download file, dbmlsync performs special checks on the last download timestamp, next last download timestamp, download file creation time, and transaction log.

Last download timestamp and next last download timestamp

Each download file contains all changes to be downloaded that occurred on the consolidated database between the file’s last download timestamp, and

its next last download timestamp. Both times are expressed in terms of the time at the consolidated database. By default the file's last download time is Jan 1, 1900 12:00 AM and the file's next last download timestamp is the time the download file was created. These defaults can be overridden by implementing the `modify_last_download_timestamp` and `modify_next_last_download_timestamp` scripts on the consolidated database.

A remote site can apply a download file only if the file's last download timestamp is less than or equal to the remote's last download timestamp. This ensures that a remote never misses operations that occur on the consolidated database. Usually when a file-based download fails based on this check, the remote has missed one or more download files. The situation can be corrected by applying the missing download files or by performing a full or download-only synchronization.

In addition, a remote site can apply a download file only if the file's next last download timestamp is greater than the remote's last download timestamp. The remote's last download timestamp is the time (at the consolidated database) up to which the remote has received all changes that are to be downloaded. The remote database's last download time is updated each time the remote successfully applies a download (normal or file-based). This check ensures that a download file will not be applied if more recent data has already been downloaded. A common case where this could happen occurs when download files are applied out of order. For example, suppose a download file F1.df is created, and another file F2.df is created later. This check ensures that F1.df cannot be applied after F2.df, because that could allow newer data in F2.df to be overwritten with older data in F1.df.

When a file-based download fails based on the next last download timestamp, no additional action is required other than to delete the file. Synchronization will succeed once a new file is received.

Creation time

The download file's creation time indicates the time at the consolidated database when creation of the file began. A download file can only be applied if the file's creation time is greater than the remote database's last upload time. The remote's last upload time is the time at the consolidated database when the remote's last successful upload was committed. This check ensures that data that has been uploaded after the creation of the download (and hence is newer than the download) will not be overwritten by older data in the download file.

When a download file is rejected based on this check, no action is required. The remote site should be able to apply the next download file.

When an upload fails because dbmlsync sent an upload to the MobiLink

synchronization server but got no acknowledgement, the remote database's last upload time may be incorrect. In this case, the creation time check cannot be performed and the remote is unable to apply download files until it completes a normal synchronization.

Transaction log

Before applying a download file, dbmlsync scans the remote database's transaction log and builds up a list of all changes that must be uploaded. Dbmlsync will only apply a download file if it does not contain any operations that affect rows with changes that must be uploaded.

MobiLink generation numbers

Generation numbers provide a mechanism for forcing remote databases to upload data before applying any more download files. This is especially useful when a problem on the consolidated database has resulted in data loss and you must recover lost data from the remote databases.

On the remote database, a separate generation number is automatically maintained for each subscription. On the consolidated database, the generation number for each subscription is determined by the `begin_publication` script. Each time a remote performs a successful upload, it updates the remote generation number with the value set by the `begin_publication` script in the consolidated database.

Each time a download file is created, the generation number set by the `begin_publication` script is stored in the download file. A remote site will only apply a download file if the generation number in the file is equal to the generation number stored in the remote database.

Note

Whenever the generation number generated by the `begin_publication` script for a file-based download changes, the remote databases must perform a successful upload before they can apply any new download files.

The `sp_hook_dbmlsync_validate_download_file` stored procedure can be used to override the default checking of the generation number.

For more information about managing MobiLink generation numbers, see:

- ◆ “`begin_publication` connection event” [*MobiLink Synchronization Reference*, page 118]
- ◆ “`end_publication` connection event” [*MobiLink Synchronization Reference*, page 155]
- ◆ “`sp_hook_dbmlsync_validate_download_file`” [*MobiLink Synchronization Reference*, page 295]

Custom validation

You can create custom validation logic to determine if a download file should be applied to a remote database. You do this with the `sp_hook_dbmlsync_validate_download_file` stored procedure. With this stored procedure, you can both reject a download file and override the default checking of the generation number.

You can use the `dbmlsync -be` option to embed a string in the file. You use the `-be` option against the file-definition database when you create the download file. This string is passed to the `sp_hook_dbmlsync_validate_download_file` through the `#hook_dict` table, and can be used in your validation logic.

☞ For more information, see “`sp_hook_dbmlsync_validate_download_file`” [*MobiLink Synchronization Reference*, page 295].

Examples

This section contains two very simple examples. Each sets up a file-based download synchronization using a consolidated database with only one table. The first is a snapshot example and the second is a timestamp-based example.

Snapshot example

This example implements file-based download for snapshot synchronization.

The following commands create the three databases used in the example: a consolidated database, a remote database, and a file-definition database.

```
dbinit scon.s.db
dbinit sremote.db
dbinit sfdef.db
```

The following commands start the three databases, create a data source name for MobiLink to use to connect to the consolidated database, and start the MobiLink synchronization server.

```
dbeng9 -n sfdef_eng sfdef.db
dbeng9 -n scon_eng scon.s.db
dbeng9 -n sremote_eng sremote.db
dbdsn -y -w fbd_demo -c "eng=scon_eng;dbf=scon.s.db;uid=dba;
pwd=sql;astart=off;astop=off"
start dbmlsrv9 -v+ -c "dsn=fbd_demo"
-zu+ -ot scon.txt
```

Create the snapshot example consolidated database

In this example, the consolidated database has one table, called T1. After connecting to the consolidated database, you can run the following SQL to create table T1:

```
CREATE TABLE T1 (
    pk    INTEGER PRIMARY KEY,
    cl    INTEGER
);
```

The following code creates a download script for the “filebased” script version:

```
CALL ml_add_table_script( 'filebased',
    'T1', 'download_cursor',
    'SELECT pk, cl FROM T1' );
```

The following code creates upload and download scripts for the “normal” script version:

```

CALL ml_add_table_script ( 'normal', 'T1',
    'upload_insert',
    'INSERT INTO T1 VALUES (?,?)');
CALL ml_add_table_script( 'normal', 'T1',
    'upload_update',
    'UPDATE T1 SET c1 = ? WHERE pk = ? ' );

CALL ml_add_table_script( 'normal', 'T1',
    'upload_delete',
    'DELETE FROM T1 WHERE pk = ?' );

CALL ml_add_table_script( 'normal', 'T1',
    'download_cursor',
    'SELECT pk, c1 FROM T1' );

COMMIT;

```

The following command creates the stored procedure `begin_pub` and specifies that `begin_pub` is the `begin_publication` script for both the “normal” and “filebased” script versions:

```

CREATE PROCEDURE begin_pub (
    INOUT generation_num integer,
    IN    username          varchar(128),
    IN    pubname           varchar(128) )
BEGIN
    SET gnum=1;
END;

CALL ml_add_connection_script(
    'filebased',
    'begin_publication',
    '{ call begin_pub( ?, ?, ? ) }' );

CALL ml_add_connection_script( 'normal',
    'begin_publication',
    '{ call begin_pub( ?, ?, ? ) }' );

```

Create the snapshot
example remote
database

In this example, the remote database also contains one table, called T1. Connect to the remote database and run the following SQL to create the table T1, a publication called P1, and a user called U1. The SQL also creates a subscription for U1 to P1.

```

CREATE TABLE T1 (
    pk    INTEGER PRIMARY KEY,
    c1    INTEGER
);

CREATE PUBLICATION P1 (
    TABLE T1
);

```



```
CREATE SYNCHRONIZATION USER U1;

CREATE SYNCHRONIZATION SUBSCRIPTION
TO P1
FOR U1;
```

The following code creates an `sp_hook_dbmlsync_validate_download_file` hook to implement user-defined validation logic in the remote database:

```
CREATE PROCEDURE sp_hook_dbmlsync_validate_download_file()
BEGIN
    DECLARE udata    varchar(256);
    SELECT value
    INTO udata
    FROM #hook_dict
    WHERE name = 'user data';
    IF udata <> 'ok' THEN
        UPDATE #hook_dict
        SET value = 'FALSE'
        WHERE name = 'apply file';
    END IF;
END
```

Create the snapshot
example file-definition
database

A file-definition database is required in MobiLink systems that use file-based downloads. This database has the same schema as the remote databases being updated by file-based download, and it contains no data or state information. The file-definition database is used solely to define the structure of the data that is to be included in the download file. One file-definition database can be used for many groups of remote databases, each defined by its own MobiLink group user name.

The following code defines the file-definition database for this sample. It creates a schema that is identical to the remote database, and also creates:

- ◆ a publication called P1 that publishes all rows of the T1 table. The same publication name must be used in the file-definition database and the remote databases.
- ◆ a MobiLink user called G1. This user represents all the remotes that are to be updated in the file-based download.
- ◆ a subscription to the publication

You must connect to `sfdef.db` before running this code.

```
CREATE TABLE T1 (
    pk    INTEGER PRIMARY KEY,
    c1    INTEGER
);
```

```

CREATE PUBLICATION P1 (
    TABLE T1
);

CREATE SYNCHRONIZATION USER G1;

CREATE SYNCHRONIZATION SUBSCRIPTION
TO P1
FOR G1;

```

Prepare for initial synchronization

To prepare your new remote database so that you can apply a download file, you need to either perform a normal synchronization or create the download file with the `dbmlsync -bg` option. This example shows you how to initialize your new remote database by performing a normal synchronization.

You can perform an initial synchronization of the remote database with the script version called `normal` that was created earlier:

```

dbmlsync -c "uid=dba;pwd=sql;eng=sremote_eng;
dbf=sremote.db" -v+ -e "sv=normal"

```

Demonstrate the snapshot example file-based download

Connect to the consolidated database and insert some data that will be synchronized by file-based download, such as the following:

```

INSERT INTO T1 VALUES( 1, 1 );
INSERT INTO T1 VALUES( 2, 4 );
INSERT INTO T1 VALUES( 3, 9 );
COMMIT;

```

The following command must be run on the computer that holds the file-definition database. It does the following:

- ◆ the `dbmlsync -bc` option creates the download file, and names it `file1.df`.
- ◆ the `-be` option includes the string “OK” in the download file that will be accessible to the `sp_dbmlsync_validate_download_file` hook.

```

dbmlsync -c
"uid=dba;pwd=sql;eng=sfdef_eng;dbf=sfdef.db"
-v+ -e "sv=filebased" -bc file1.df -be ok -ot fdef.txt

```

To apply the download file, run `dbmlsync` with the `-ba` option on the remote database, supplying the name of the download file you want to apply:

```

dbmlsync -c "uid=dba;pwd=sql;eng=sremote_eng;
dbf=sremote.db" -v+ -ba file1.df -ot remote.txt

```

The changes are now applied to the remote database. Open Interactive SQL, connect to the remote database, and run the following SQL command to verify that the remote has the data:

```

SELECT * FROM T1

```

Clean up the snapshot example

The following commands stop all three database engines and erase the files.

```
del file1.df
dbmlstop -h -w
dbstop -y -c eng=sfdef_eng
dbstop -y -c eng=scons_eng
dbstop -y -c eng=sremote_eng
dberase -y sfdef.db
dberase -y scons.db
dberase -y sremote.db
```

Timestamp-based example

This example implements file-based download for timestamp-based synchronization.

The following commands create the three databases used in the example: a consolidated database, a remote database, and a file-definition database.

```
dbinit tcons.db
dbinit tremote.db
dbinit tfdef.db
```

The following commands start the three databases, create a data source name for MobiLink to use to connect to the consolidated database, and start the MobiLink synchronization server.

```
dbeng9 -n tfdef_eng tfdef.db
dbeng9 -n tcons_eng tcons.db
dbeng9 -n tremote_eng tremote.db
dbdsn -y -w tfbd_demo -c "eng=tcons_eng;dbf=tcons.db;uid=dba;
pwd=sql;astart=off;astop=off"
start dbmlsrv9 -v+ -c "dsn=tfbd_demo" -zu+ -ot tcons.txt
```

Create the timestamp example consolidated database

In this example, the consolidated database has one table, called T1. After connecting to the consolidated database, you can run the following code to create T1:

```
CREATE TABLE T1 (
  pk      INTEGER PRIMARY KEY,
  c1      INTEGER,
  last_modified  TIMESTAMP DEFAULT TIMESTAMP
);
```

The following code defines a script version called normal with a minimal number of scripts. This script version is used for synchronizations that do *not* use file-based download.

```

CALL ml_add_table_script( 'normal', 'T1',
    'upload_insert',
    'INSERT INTO T1( pk, c1) VALUES( ?, ? )' );

CALL ml_add_table_script( 'normal', 'T1',
    'upload_update',
    'UPDATE T1 SET c1 = ? WHERE pk = ? ' );

CALL ml_add_table_script( 'normal', 'T1',
    'upload_delete',
    'DELETE FROM T1 WHERE pk = ? ' );

CALL ml_add_table_script( 'normal', 'T1',
    'download_cursor',
    'SELECT pk, c1 FROM T1 WHERE last_modified >= ?' );

```

The following code sets the generation number for all subscriptions to 1. It is good practice to use generation numbers in case your consolidated database ever becomes lost or corrupted and you need to force an upload.

```

CREATE PROCEDURE begin_pub (
    INOUT generation_num integer,
    IN     username          varchar(128),
    IN     pubname           varchar(128) )
BEGIN
    SET generation_num = 1;
END;

CALL ml_add_connection_script( 'normal',
    'begin_publication',
    '{ call begin_pub( ?, ?, ? ) }' );

COMMIT;

```

The following code defines the script version called filebased. This script version is used to create file-based downloads.

```

CALL ml_add_connection_script( 'filebased',
    'begin_publication',
    '{ call begin_pub( ?, ?, ? ) }' );

CALL ml_add_table_script( 'filebased', 'T1',
    'download_cursor',
    'SELECT pk, c1 FROM T1 WHERE last_modified >= ?' );

```

The following code sets the last download time so that all changes that occurred within the last five days will be included in download files. Any remote that has missed all the download files created in the last five days will have to perform a normal synchronization before being able to apply any more file-based downloads.

```
CREATE PROCEDURE ModifyLastDownloadTimestamp(  
    INOUT last_download_timestamp TIMESTAMP,  
    IN    ml_username                VARCHAR(128) )  
BEGIN  
    SELECT dateadd( day, -5, CURRENT_TIMESTAMP )  
    INTO last_download_timestamp;  
END;  
  
CALL ml_add_connection_script( 'filebased',  
    'modify_last_download_timestamp',  
    'CALL ModifyLastDownloadTimestamp( ?, ? )' );  
  
COMMIT;
```

Create the timestamp
example remote
database

In this example, the remote database also contains one table, called T1. After connecting to the remote database, run the following code to create table T1, a publication called P1, and a user called U1. The code also creates a subscription for U1 to P1.

```
CREATE TABLE T1 (  
    pk    INTEGER PRIMARY KEY,  
    cl    INTEGER  
);  
  
CREATE PUBLICATION P1 (  
    TABLE T1  
);  
  
CREATE SYNCHRONIZATION USER U1;  
  
CREATE SYNCHRONIZATION SUBSCRIPTION  
TO P1  
FOR U1;
```

The following code defines a `sp_hook_dbmlsync_validate_download_file` stored procedure. This stored procedure prevents the application of download files that do not have the string “ok” embedded in them.

```

CREATE PROCEDURE sp_hook_dbmlsync_validate_download_file()
BEGIN
    DECLARE udata    varchar(256);

    SELECT value
    INTO udata
    FROM #hook_dict
    WHERE name = 'user data';

    IF udata <> 'ok' THEN
        UPDATE #hook_dict
        SET value = 'FALSE'
        WHERE name = 'apply file';
    END IF;
END

```

Create the timestamp example file-definition database

The following code defines the file-definition database for the timestamp example. It creates a table, a publication, a user, and a subscription for the user to the publication.

```

CREATE TABLE T1 (
    pk    INTEGER PRIMARY KEY,
    c1    INTEGER
);

CREATE PUBLICATION P1 (
    TABLE T1
);

CREATE SYNCHRONIZATION USER G1;

CREATE SYNCHRONIZATION SUBSCRIPTION
TO P1
FOR G1;

```

Prepare for initial synchronization

To prepare your new remote database so that you can apply a download file, you need to either perform a normal synchronization or create the download file with the dbmlsync -bg option. This example shows you how to use -bg.

The following code defines a script version called filebased_init for the consolidated database. This script version has a single begin_publication script.

```

CALL ml_add_table_script(
    'filebased_init', 'T1', 'download_cursor',
    'SELECT pk, c1 FROM T1' );

CALL ml_add_connection_script(
    'filebased_init',
    'begin_publication',
    '{ call begin_pub( ?, ?, ? ) }' );

COMMIT;

```

The following two command lines create and apply an initial download file using the script version called `filebased_init` and the `-bg` option.

```
dbmlsync -c "uid=dba;pwd=sql;eng=tfdef_eng;dbf=tfdef.db"
-v+ -e "sv=filebased_init" -bc tfile1.df -be ok -bg
-ot tfdef1.txt
```

```
dbmlsync -c "uid=dba;pwd=sql;eng=tremote_eng;dbf=tremote.db"
-v+ -ba tfile1.df -ot tremote.txt
```

Demonstrate the
timestamp example
file-based download

Connect to the consolidated database and insert some data that will be synchronized by file-based download, such as the following:

```
INSERT INTO T1(pk, c1) VALUES( 1, 1 );
INSERT INTO T1(pk, c1) VALUES( 2, 4 );
INSERT INTO T1(pk, c1) VALUES( 3, 9 );
commit;
```

The following command line creates a download file containing the new data.

```
dbmlsync -c
"uid=dba;pwd=sql;eng=tfdef_eng;dbf=tfdef.db"
-v+ -e "sv=filebased" -bc tfile2.df -be ok -ot tfdef2.txt
```

The following command line applies the download file to the remote database.

```
dbmlsync -c "uid=dba;pwd=sql;eng=tremote_eng;dbf=tremote.db"
-v+ -ba tfile2.df -ot tfdef3.txt
```

The changes are now applied to the remote database. Open Interactive SQL, connect to the remote database, and run the following SQL command to verify that the remote has the data:

```
SELECT * FROM T1
```

Clean up the timestamp
example

The following commands stop all three database engines and then erase the files.

```
del file1.df
dbmlstop -h -w
dbstop -y -c eng=tfdef_eng
dbstop -y -c eng=tcons_eng
dbstop -y -c eng=tremote_eng
dberase -y tfdef.db
dberase -y tcons.db
dberase -y tremote.db
```


CHAPTER 7

Server-Initiated Synchronization

About this chapter

Server-initiated synchronization allows you to 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. This MobiLink component provides programmable options for determining what changes in the consolidated database will initiate synchronization, how remotes are chosen to receive push messages, and how the remotes respond.

This chapter describes how to develop a server-initiated synchronization application, documents a Software Development Kit that you can use to add support for new devices, and provides information about several sample applications.

Contents

Topic:	page
Introduction	138
Supported platforms	141
Setting up server-initiated synchronization	142
Push requests	143
Set up the Notifier	145
Set up the Listener	154
Listener Software Development Kit	162
Deployment considerations	163
Walkthrough of server-initiated synchronization	164
Sample applications	166

Introduction

MobiLink server-initiated synchronization works as follows:

- ◆ **Push requests** cause synchronization to occur. A push request takes the form of some data that you insert into a table, or in some cases data inserted into a temporary table or even just a SQL result set. You can create push requests in any way that you cause data to be inserted into a table. For example, a push request could be created by a database trigger that is activated when a price changes. Any database application can create push requests, including the Notifier.

☞ For more information, see [“Push requests” on page 143](#).

- ◆ **The Notifier** is a Java program running on the same computer as the MobiLink synchronization server. It polls the consolidated database on a regular basis, looking for push requests. You control how often the Notifier polls the database. You specify business logic that the Notifier uses to gather push requests, including which remote devices should be notified. When the Notifier detects a request, it sends the message associated with the request via SMTP or UDP to a Listener on one or more remote devices. You have the option to send repeatable messages with an expiry time.

To set up Notifiers, you edit a properties file. The properties file is a text file that includes configuration information, including the query that the Notifier uses when it gathers push requests. A properties file can configure multiple Notifiers.

To run Notifiers, you use the `dbmlsrv9 -notifier` option.

☞ For more information, see [“Set up the Notifier” on page 145](#).

- ◆ **The Listener** is a program that is installed on each remote device. It receives messages from the Notifier and initiates action. The action is most frequently synchronization, but can be other things. You can configure the Listener to act only on messages from selected sources, or with specific content.

You can send messages from the Notifier to the Listener on an SMTP gateway or a UDP gateway. When you use an SMTP gateway, you send an e-mail message that your carrier converts into SMS before the Listener receives it. Most carriers provide an e-mail-to-SMS service.

On Windows or Windows CE, the Listener is an executable program configured by command line options. In order to receive a message, the remote device must be on and the Listener must be started.

On the Palm OS, you first create a configuration file by running the Listener program with special command line settings on a Windows

machine. Then you copy the configuration file to your Palm device.

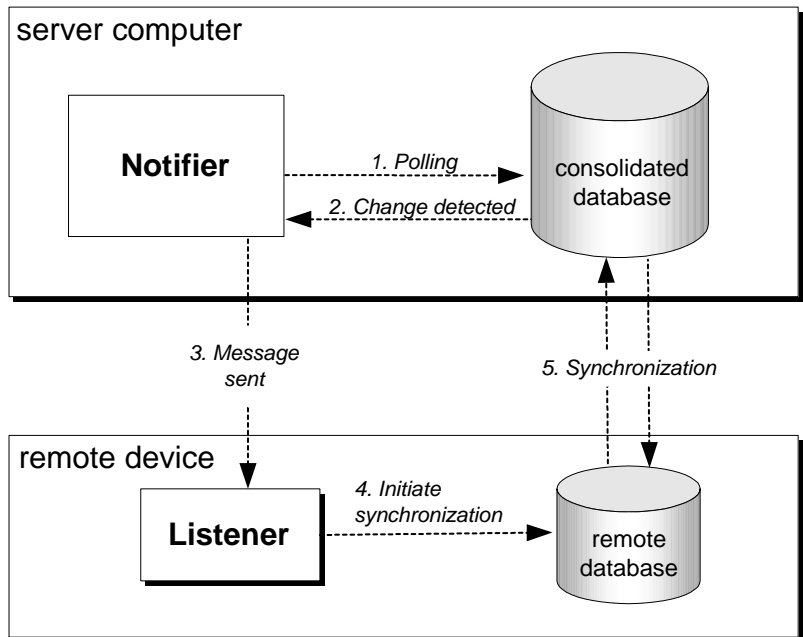
☞ For more information, see [“Set up the Listener” on page 154](#).

Example

For example, a fleet of truck drivers uses mobile databases to determine routes and delivery points. A driver synchronizes a report of a traffic disruption. The Notifier detects the change in the consolidated database and automatically sends a message to the remote device of every driver whose route is affected, which causes the drivers’ remote databases to synchronize so that the drivers will use an alternate route.

The notification process

In the following illustration, the Notifier polls a consolidated database and detects a change that it has been configured to look for. In this scenario, the Notifier sends a message to a single remote device, resulting in the remote database being updated via synchronization.



Following are the steps that occur in this example:

1. Using a query based on business logic, the Notifier polls the consolidated database to detect any change that needs to be synchronized to the remote.
2. When a change is detected, the Notifier prepares a message to send to the remote device.
3. The Notifier sends a message over UDP or SMTP. Most

telecommunication providers have a service that lets you send SMS messages by sending e-mails via SMTP to a special address.

4. The Listener checks the contents and sender of the message against a filter.
5. If the message matches the filter, the Listener runs a program that has been associated with the filter. For example, the Listener runs dbmlsync or it launches an UltraLite application.

Supported platforms

If you use the Notifier on UNIX, the computer must have JRE 1.4.1 or higher. If you are targeting Palm remotes, you must have at least one Windows device or laptop to create a configuration file using the Listener utility, `dblsn`.

- ◆ **SMS messages** can be transmitted through an SMTP gateway and go through an e-mail-to-SMS conversion that is provided by wireless carriers. This has been tested on the following platforms:
 - Palm 3.5 and higher on the Treo 180 and the Kyocera 6035
 - Pocket PC 2002 with Sierra Wireless AirCard 510, 555, 710, or 750
 - Windows 2000 and XP with the Sierra Wireless AirCard 510, 555, 710, or 750
- ◆ **UDP messages** have been tested on the following platforms:
 - Pocket PC 2002
 - Windows 2000 and XP

Setting up server-initiated synchronization

To set up server-initiated synchronization, you should perform the following steps. This assumes that synchronization is already set up.

1. Set up push requests.
See [“Push requests” on page 143](#).
2. Set up the Notifier.
See [“Set up the Notifier” on page 145](#).
3. Set up the Listener.
See [“Set up the Listener” on page 154](#).

Push requests

The Notifier sends a message to a remote database when it detects a push request. A push request is a row in a SQL result set that contains the following columns in the following order. The first five columns are required and the last two columns are optional.

Column	Description
request id	INTEGER. A unique ID for a push request.
gateway	VARCHAR. The gateway on which to send the message. The names of gateways are defined in your Notifier properties file.
subject	VARCHAR. The subject line of the message.
content	VARCHAR. The content of the message.
address	VARCHAR. Destination address. The format of the address is gateway-specific. For example, in an SMTP gateway the address is an e-mail address.
resend interval	INTEGER. Optional. How often the message should be resent, in minutes. Useful especially when the network is unreliable or the remote device may go out of coverage. The Notifier assumes that all attributes associated with a resendable notification request do not change: subsequent updates are ignored after the first poll of the request. The Notifier automatically adjusts the next polling interval if a resendable notification must be sent before the next polling time. You can stop a resendable notification using the request_cursor query or by deleting the request from the request table. The default is to send exactly once, with no resend.
time to live	INTEGER. Optional. The time in minutes until the request should be deleted. If this value is 0, NULL, or not specified, the default is to send exactly once, with no resend.

A push request occurs when the Notifier finds a new row in the result set. The Notifier uses this result set to decide what messages to send to specific addresses. Each row of the result set represents one push request to send one message to one address through one gateway.

In a typical implementation, you add a table to your consolidated database with the columns listed above.

Create push requests

There are many ways that you can create push requests. You can use any method for inserting data. Following is a list of common ways to create push requests:

- ◆ Define a database trigger. For example, create a trigger that detects when a price changes and then inserts push request data into a table of push requests.
- ◆ Use a database client application that inserts data into a push request table directly.
- ◆ Use the Notifier property `begin_poll`.
 - ☞ For more information, see `begin_poll` in [“Notifier properties” on page 147](#).
- ◆ Manually insert push request data.

Detect push requests

The Notifier obtains a set of push requests by executing a SQL query that you provide in the `request_cursor` property. The query contains your business logic for determining who gets the message, what the message contains, and when the message is sent.

☞ For more information about querying the consolidated database, see `request_cursor` in [“Notifier properties” on page 147](#).

Delete push requests

You should implement a cleanup system to delete push requests or you may flood your system. The most straightforward way to do this is to use the Notifier property `request_delete`. This property is a SQL statement with a request ID as a parameter.

☞ For more information about the `request_delete` property, see [“Notifier properties” on page 147](#).

Set up the Notifier

You start Notifiers on the dbmlsrv9 command line. You configure Notifiers with a Notifier properties file.

Start the Notifier

The Notifier is implemented as a MobiLink startup Java class. It runs as a thread within the Java VM. The Notifier thread polls the consolidated database and sends messages according to your business logic.

To start the Notifier, use the dbmlsrv9 option **-notifier**. Optionally, you can also specify the name of your Notifier properties file.

Following is a partial dbmlsrv9 command line:

```
dbmlsrv9 ... -notifier c:\config.notifier
```

Configure the Notifier

You configure Notifiers using a Notifier properties file. This text file can have any name. The easiest way to create this file is to alter the template, *%asany%\samples\MobiLink\template.notifier*. The properties are described in the following sections.

You can have several Notifier property files. To identify the properties file you want to use, specify the name and location when you start dbmlsrv9 with the -notifier option. Following is a partial dbmlsrv9 command line:

```
dbmlsrv9 ... -notifier "c:\samples\CarDealer.notifier"
```

If you do not specify a file with the -notifier option, by default the Notifier looks for a file called *config.notifier* in your system path. If it doesn't find one, the MobiLink synchronization server issues an error and fails to start.

A Notifier properties file can configure and start multiple Notifiers and multiple gateways. You provide a name for each Notifier and gateway that you want to define.

Notifier properties must be entered on one line, but you can use the backslash (\) as a line continuation character. The backslash is also an escape character.

You can use the following escape sequences in your property settings:

Escape sequence	Description
\b	\u0008: backspace BS
\t	\u0009: horizontal tab HT
\n	\u000a: linefeed LF
\f	\u000c: form feed FF
\r	\u000d: carriage return CR
\"	\u0022: double quote "
\'	\u0027: single quote '
\\	\u005c: backslash \
\uhhhh	Unicode
\xhh	\xhh: ASCII escape
\e	\u001b: Unicode escape

The properties file contains four sections:

- ◆ Common properties
- ◆ Notifier properties
- ◆ SMTP gateway properties (optional)
- ◆ UDP gateway properties (optional)

Common properties

There is one common property, verbosity.

verbosity

Verbosity affects all Notifiers and gateways in the properties file. You can set the verbosity to the following levels:

Level	Description
0	No trace (the default)
1	Startup, shutdown, and property trace
2	Display notification messages
3	Poll-level trace

For example,

```
verbosity=2
```

Notifier properties

The following properties can be set in the Notifier properties file. The `enable` and `request_cursor` properties are required. All other Notifier properties are optional.

You can have multiple Notifiers running with one MobiLink server. To set up additional Notifiers, copy the properties for one Notifier and provide a different Notifier name and property values.

enable Specify `enable=yes` to use a Notifier. You can define and use multiple Notifiers in one file.

For example, a Notifier called `NotifierA` is enabled with the following line:

```
Notifier(NotifierA).enable=yes
```

isolation Isolation is an optional property that controls the isolation level of the Notifier's database connection. The default value is 1. You can use the following values:

Value	Isolation level
0	Read uncommitted
1	Read committed (the default)
2	Repeatable read
3	Serializable

For example, the isolation level is set for `NotifierA` with the following line:

```
Notifier(NotifierA).isolation=2
```

connect_string By default, the Notifier uses `ianywhere.ml.script.ServerContext` to connect to the consolidated database. This means that it uses the connection string that was specified in the current `dbmlsrv9` session's command line.

This is an optional property that can be used to override the default connection behavior. It is a JDBC connection string. You can use it to connect to any database, including the consolidated database. It may be useful to connect to another database when you want notification logic and data to be separate from your synchronization data.

☞ For more information, see [“ServerContext interface” on page 248](#).

For example, a Notifier called `Simple` is configured to use a DSN with the

following line:

```
Notifier(Simple).connect_string = dsn=SYS_DB\  
;uid=user;pwd=myPwd
```

poll_every

This property specifies the polling interval. If no unit is specified, the interval is in seconds. You can specify S, M, and H for units of seconds, minutes, and hours. You can also combine units, as in 1H 30M 10S.

If the Notifier loses the database connection, it will recover automatically after the polling interval.

This property is optional. The default is 30 seconds.

For example, a Notifier called Simple is configured to poll every three hours with the following line:

```
Notifier(Simple).poll_every = 3H
```

gui

This controls whether the Notifier dialog is open on the computer where the Notifier is running. This user interface allows users to temporarily change the polling interval, or poll immediately. It can also be used to shut down the Notifier without shutting down the MobiLink synchronization server. (Once stopped, the Notifier can only be restarted by shutting down and restarting the MobiLink synchronization server.)

This property is optional. The default is ON.

For example, the Notifier dialog is disabled for a Notifier called Simple with the following line:

```
Notifier(Simple).gui=off
```

begin_connection

This is a SQL statement that runs as a separate transaction after the Notifier connects to the database and before the first poll. For example, this property can be used to create temporary tables or variables.

If the Notifier loses its connection to the consolidated database, it will re-execute this transaction immediately after reconnecting.

You should not use this property to change isolation levels. To control isolation levels, use the isolation property.

For example, begin_connection is defined for a Notifier called Car Dealer with the following line. The backslash is a line continuation character.

```
Notifier(Car Dealer).begin_connection = \  
set temporary option blocking = 'off'
```

begin_poll

This is a SQL statement that is executed before each poll. Typical uses are to detect data change in the database and create push requests.

The statement is executed in a standalone transaction.

This property is optional. The default is NULL.

For example, the following SQL statement inserts rows into a table called PushRequest. Each row in this table represents a message to send to an address. The WHERE clause determines what push requests are inserted into the PushRequest table. The backslash is used as a line continuation character.

```
Notifier(NotifierA).begin_poll = \
INSERT INTO PushRequest \
( gateway, mluser, subject, content ) \
  SELECT 'MyGateway', DISTINCT mluser, \
    'sync', stream_param \
  FROM MLUserExtra, mluser_union, Dealer \
  WHERE
    MLUserExtra.mluser = mluser_union.name \
  AND( push_sync_status = 'waiting for request' \
    OR datediff( hour, last_status_change, \
      now() ) > 12 ) \
  AND ( mluser_union.publication_name is NULL \
    OR mluser_union.publication_name = \
      'FullSync' ) \
  AND \
    Dealer.last_modified > mluser_union.last_sync_time
```

shutdown_query

This is a SQL statement that is executed right after begin_poll. The result should contain only the value yes (or 1) or no (or 0). To shut down the Notifier, specify yes or 1. This statement is executed as a standalone transaction.

If you are storing the connection state in a table, then you can use the end_connection property to reset the state before the Notifier disconnects.

For example, shutdown_query is defined for a Notifier called Simple with the following line. The backslash is a line continuation character.

```
Notifier(Simple).shutdown_query = \
  SELECT COUNT(*) FROM NotifierShutdown \
  WHERE name='Simple'
```

request_cursor

This property determines what information is sent in the message, who receives the information, when, and where. You must set this property.

This property specifies a SQL statement for the Notifier to collect push requests from the consolidated database. The result set of this statement must contain five columns, and can optionally contain two other columns. These columns can have any name, but must be in the following order in the result set:

- ◆ request id
- ◆ gateway
- ◆ subject
- ◆ content
- ◆ address
- ◆ resend interval
- ◆ time to live

☞ For more information about these columns, see [“Push requests” on page 143](#).

You might want to include a WHERE clause in your request_cursor to filter out requests that have been satisfied. For example, you can add a column to your push request table to track the time you inserted a request, and then use a WHERE clause to filter out requests that were inserted prior to the last time the user synchronized.

The statement is executed in a standalone transaction.

Following is an example of a request_cursor. The backslash is a line continuation character.

```
Notifier(Simple).request_cursor = \
SELECT \
    PushRequest.req_id, MLUserExtra.gateway, \
    PushRequest.subject, \
    PushRequest.content, MLUserExtra.address, \
    PushRequest.resend_minute, PushRequest.minute_to_live \
FROM PushRequest, mluser_union, MLUserExtra \
WHERE PushRequest.mluser = mluser_union.name \
AND PushRequest.mluser = MLUserExtra.mluser \
AND PushRequest.req_time > mluser_union.last_sync_time
```

request_delete

This is a SQL statement that specifies cleanup operations. The statement takes the request id as its only parameter. The placeholder for a parameter is a question mark (?).

Using the delete statement, the Notifier can automatically remove two forms of old request:

- ◆ **implicitly dropped requests** requests that appeared previously, but disappeared from the current set of requests obtained from the query.
- ◆ **expired requests** requests that have expired based on their resend attributes and the current time. Requests without resend attributes are considered expired even if they appear in the next request.

You can write the WHERE clause of request_delete in such a way that previous requests that have been handled by the Listener will not enter the next set of requests. For example, the Car Dealer sample uses request time and last synchronization time. This not only stops further messages for the same request, but also allows the Notifier to delete the implicitly dropped request.

This property is optional if you have provided another process to do the cleanup.

The statement is executed per request ID in a standalone transaction.

For example, the Notifier called Simple is configured to substitute a req_id previously obtained from request_cursor for the question mark (?):

```
Notifier(Simple).request_delete = \  
DELETE FROM PushRequest WHERE req_id = ?
```

end_poll

This is a SQL statement that is executed after each poll. Typical uses are to perform customized cleanup or track polling.

The statement is executed in a standalone transaction.

This property is optional. The default is NULL.

For example,

```
Notifier(Simple).end_poll = call reportAliveRequests( )
```

end_connection

This is a SQL statement that runs as a separate transaction just before a Notifier database connection is closed. For example, this property can be used to delete temporary storage such as SQL variables and temporary tables.

The statement is executed in a standalone transaction.

For example, end_connection is defined for the Simple Notifier with the following line. The backslash is a line continuation character.

```
Notifier(Simple).end_connection = \  
DELETE FROM NotifierShutdown WHERE name = 'Simple'
```

SMTP gateway properties

SMTP gateway configuration is required only if you are using an SMTP gateway.

SMTP gateways can be used to send e-mail messages. They can also send SMS messages to SMS listeners via a wireless carrier's e-mail-to-SMS service.

The following properties can be set in the Notifier properties file. The enable and server properties are required. The user and password properties may be required, depending on your SMTP server setup. All other SMTP gateway properties are optional.

You can have multiple SMTP gateways. To set up additional SMTP gateways, copy the properties for one gateway and provide a different gateway name and property values.

enable

Specify enable=yes to use an SMTP gateway. You can define and use multiple SMTP gateways in one file.

For example, an SMTP gateway called Gate2 is enabled with the following line:

```
SMTP(Gate2).enable = yes
```

server

This is the IP address of the SMTP server for sending the message to the Listener.

For example,

```
SMTP(Gate2).server = mail.mycorp.com
```

user

This is the user name for your SMTP service. Your SMTP service may not require a user name.

For example,

```
SMTP(Gate2).user = smtp_username
```

password

This is the password for your SMTP service. Your SMTP service may not require a password.

For example,

```
SMTP(Gate2).password = smtp_password
```

sender

This is the sender address of the e-mails (SMTP requests).

The sender may or may not be available as an action variable to the Listener.

For example,

```
SMTP(Gate2).sender = SimpleNotifier@mycorp.com
```

UDP gateway properties

UDP gateway configuration is required only if you are using a UDP gateway.

UDP is useful for development and for applications over wireless LANs.

The format of the UDP message is [*subject*] *content* where *subject* and *content* come from the subject and content columns of the request_cursor Notifier property.

The following properties can be set in the Notifier properties file. The enable property is required. All other UDP gateway properties are optional.

You can have multiple UDP gateways. To set up additional UDP gateways, copy the properties for one gateway and provide a different gateway name and property values.

enable

Specify enable=yes to use a UDP gateway. You can define and use multiple UDP gateways in one file.

For example, a UDP gateway called Gate3 is enabled with the following line:

```
UDP(Gate3).enable = yes
```

sender

This is the IP address of the sender. This property is optional, and is only required for multi-homed hosts. The default is localhost.

For example,

```
UDP(Gate3).sender = \  
my_server_on_an_alternate_network_card.mycorp.com
```

sender_port

This is the port to use for sending the UDP packet. This property is optional; you may need to set it if your firewall restricts outgoing traffic. If not set, your operating system will assign a free port.

For example,

```
UDP(Gate3).sender_port = 1234
```

listener_port

This is the port on the remote device where the gateway sends the UDP packet. This property is optional. The default is the default listening port of the supplied UDP Listener (5001).

For example,

```
UDP(Gate3).listener_port = 4321
```

Set up the Listener

The Listener runs on remote devices. It receives messages from the Notifier and processes them into actions. For example,

```
dblsn -l message=[FullSync]Host=myML.com;action=run dbmlsync.exe
...
```

The message that you supply acts as a filter. In this example, the Listener will only start dbmlsync if it receives a message with the subject “FullSync” and contents “Host=myML.com”. This message corresponds to the third and fourth columns in the result set created by the Notifier property request_cursor.

A convenient way to configure the Listener is to store the command line options in a text file and access it with the @ symbol. For example, store the settings in *mydblsn.txt* and start the Listener by typing

```
dblsn @mydblsn.txt
```

The Listener utility

Configures and starts the Listener on Windows devices, including Windows CE.

Syntax

```
dblsn [ options ] -l message-handler [ -l message-handler... ]
```

```
message-handler : [ filter;... ];action
```

```
filter :
```

```
[ message = string | message_start = string ]
```

```
[ sender = string ]
```

```
action :
```

```
  action = command [ ;altaction = command ]
```

```
[ ;continue = { yes | no } ]
```

```
[ ;maydial = { yes | no } ]
```

```
command :
```

```
{ start program [ program-arguments ]
```

```
  | run program [ program-arguments ]
```

```
  | post window-message to window-class
```

```
  | tcpip-socket-action
```

```
  | DBLSN FULL SHUTDOWN }
```

```

tcpip-socket-action :
socket port=app-port
[ ;host=app-host ]
[ ;sendText=text1 ]
[ ;recvText= text2 [ ;timeout=num-sec ] ]

```

Parameters

Options The following options can be used to configure the Listener. If you are creating a configuration file for a Palm device, you must use -n. The rest are optional.

dblsn options	Description
-a option	Specifies Listener dll options. To specify multiple options, repeat -a. For example, -a port=2439 -a ShowSenderPort. To see options for your dll, type: <code>dblsn -d filename.dll -a ?</code> The default dll is <code>lsn_udp.dll</code> .
-d filename	Specifies the Listener dll that you want to use. On Windows you can specify <code>lsn_swi510.dll</code> . The default is <code>lsn_udp.dll</code> .
-i seconds	Sets the polling interval in seconds. This is the frequency at which the Listener listens for messages. The default is 30 seconds.
-m	Turns on message logging. The default is off.
-n [filename]	Creates a Palm Listener configuration file. If you use -n, dblsn generates the file and then shuts down. When you use -n, options such as -a, -d, -i, -m, and -p are ignored. If you use -n but omit the <i>filename</i> , the Palm configuration file is called <code>lsncfg.pdb</code> .
-o filename	Logs output messages to a file.
-os bytes	Specifies a maximum size for the log file in bytes. The minimum size is 10 000. By default, there is no limit.
-ot filename	Logs output messages to file, but first truncates the file.
-p	Allows automatic idle power-off. This option has an effect only on CE devices. Use it to allow the device to shut down when idle. By default, the Listener prevents the device from shutting itself down.
-q	Runs in a minimized window.
-r bytes	Specifies the minimum number of bytes per record in the Palm Listener configuration file. The default is 1 000.

dblsn options	Description
-v [level]	<p>Sets the verbosity level for the dblsn log and console. The <i>level</i> can be 0, 1, 2, or 3:</p> <ul style="list-style-type: none"> ◆ 0 - show no informational messages (the default). ◆ 1 - show Listener dll messages and basic action tracing steps. ◆ 2 - show level 1 plus detailed action tracing steps. ◆ 3 - show level 2 plus polling and listening states.

-l message-handler -l allows you to specify a message handler, which is a filter-action pair. The filter determines which message should be handled, and the action is invoked when the filter matches a message. You can specify multiple instances of -l. Each instance of -l specifies a different message handler.

Filters You specify a filter to compare to an incoming message. If the filter matches, the action you specify is invoked.

There are three types of filter you can specify:

- ◆ **message** compares the entire message to text you specify. To match, this filter must also be the exact same length as the message.

The Listener translates non-printable characters to a tilde (~), so if there are non-printable characters, the pattern to match must also use tildes.

The format of messages is carrier-dependent. For example, you may want to match a message from a sender named Bob@mail.com with the subject Help and the message Me. In UDP, this would appear as [Help]Me. On Bell Mobility's e-mail to SMS conversion service, it would be Bob@mail.com[Help]Me. On Fido's e-mail to SMS conversion service, it would be sent as Bob@mail.com\n(Help)\nMe, but would be translated by the Listener to Bob@mail.com~(Help)~Me. You must test with your carrier to determine the appropriate format, using the dblsn options -v and -m.

- ◆ **message_start** compares a portion of the message (from the beginning) to text that you specify. When you specify message_start, the Listener creates the action variables \$message_start and \$message_end. For more information, see Action variables, below.

The Listener translates non-printable characters to a tilde (~) so if there are non-printable characters, the pattern match must also use tildes.

- ◆ **sender** is the sender of the message. You can only specify one sender. For UDP gateways, the sender is the IP address of the host of the gateway. For SMTP gateways, the sender is optionally specified with the sender SMTP property.

The filter is optional. If you do not specify a filter, the action is performed when any message is received.

Action and altaction Each filter is associated with an action and, optionally, an alternative action called the altaction. If a message meets the conditions of the filter, the action is invoked. You must specify an action. If you specify an altaction, the altaction is invoked only if the action fails. Palm devices do not recognize altaction.

The action and altaction are specified as { **action** | **altaction** }= *command*. For each action and altaction, there can be one command, and it can be one of **start**, **run**, **post**, **socket**, or **DBLSN FULL SHUTDOWN**.

- ◆ **start** spawns a process. When you start a program, the Listener continues listening for more messages.

The following example starts dbmlsync with some command line options, parts of which are obtained from the message.

```
"action=start dbmlsync.exe @dbmlsync.txt -n
$message_end -wc dbmlsync_$message_end -e sch=INFINITE"
```

- ◆ **run** runs the program and waits for it to finish. The Listener resumes listening after the process is complete. You cannot use run on Palm devices.

The following example runs dbmlsync with some command line options, parts of which are obtained from the message.

```
"action=run dbmlsync.exe @dbmlsync.txt -n $message_end"
```

- ◆ **post** posts a message to a window class. This is required by dbmlsync when scheduling is on. Use the dbmlsync -wc option to specify the window class name. Post is also used when signaling applications that use Windows messages. You cannot use post on Palm devices.

The following example posts a Windows message registered as dbas_synchronize to a dbmlsync instance registered with the class name dbmlsync_FullSync.

```
action=post dbas_synchronize to dbmlsync_FullSync
```

- ◆ **socket** notifies an application by making a TCP/IP connection. This is especially useful for Java and eMbedded Visual Basic applications, because Java and eVB don't support custom window messaging, and

eVB doesn't support command line parameters. You can connect to a local socket by specifying just a port, or you can connect to a remote socket by specifying the host along with the port. Using `sendText`, you can send a string. You can optionally verify that the response is as expected with `recvText`. When you use `recvText`, you can also specify a timeout to avoid hanging in the case of application or network problems.

The following example forwards the string in `$sender=$message` to a local application that is listening on port 12345, and expects the application to send back "beeperAck" as an acknowledgement.

```
-l "action='socket
port=12345;
sendText=$sender=$message;
recvText=beeperAck;
timeout=5' "
```

- ◆ **DBLSN FULL SHUTDOWN** After shutdown, the Listener stops handling inbound messages. The remote user must restart the Listener in order to continue with server-initiated synchronization. This feature is mostly useful during development.

For example, `action='DBLSN FULL SHUTDOWN'`

You can also specify the following action options:

- ◆ **continue {yes|no}** specifies whether the Listener should continue after finding the first match. This is useful when you specify multiple `-l` clauses to cause one message to initiate multiple actions. This option is not recognized by Palm devices. The default is no.
- ◆ **maydial {yes|no}** specifies whether the action can dial the modem. This provides information to the Listener to decide whether to release the modem or not before the action. This option is useful when the action or altaction need exclusive access to the modem used by the Listener. This option is not recognized by Palm devices. The default is yes.

The escape character for action and altaction is a dollar sign (\$), so to specify a single dollar sign as plain text, type \$\$.

The Listener determines that an action has failed (and then invokes the altaction) in the following ways:

- ◆ When you **run** a program, the Listener determines that the program has failed if the program doesn't exist or if it returns a non-zero return code.
- ◆ When you **start** a program, the Listener doesn't wait for a return code, so it can only tell that the action has failed if the program does not exist.

- ◆ When you **post** to a window class, the Listener determines that the action has failed if no window class has been registered with the given name or the operating system has reported that the post failed.
- ◆ When you perform a **socket** action, the Listener determines that the action has failed if it failed to connect, send, or receive expected acknowledgement before timeout.

You can only specify one action and one altaction in each instance of -l. You can also write a cover program or batch file that contains multiple actions, and run it as the action.

Following is an example of the specification of altaction. In this example, the \$message_end that matches the filter is the stream parameter for connecting to MobiLink. The primary action is to post the dbas_synchronize Windows message to the dbmlsync_FullSync window. The example uses altaction to start (not run) dbmlsync with the window class name dbmlsync_FullSync if the primary action fails. This is the standard way to make the Listener work with dbmlsync scheduling.

```
-l "message_start=[sync];
    action='post dbas_synchronize to dbmlsync_FullSync';
    altaction='start dbmlsync.exe
                @dbmlsync.txt
                -wc dbmlsync_FullSync
                -e adr=$message_end;sch=INFINITE' "
```

Action variables

The following Listener action variables can be used anywhere in the action or altaction.

An action variable is replaced just before the action or altaction is performed.

Listener action variables start with a dollar sign (\$). The escape character is also a dollar sign, so to specify a dollar sign as plain text, type \$\$\$. For example, type \$\$message_start when you don't want \$message_start to be substituted.

Action variable	Description
\$sender	The sender of the message.
\$message_start	A portion of the text of the message from the beginning, as specified in -l message_start. This variable is only available if you have specified -l message_start.
\$message	The content of the message.
\$message_end	The part of the message that is left over after the part specified in -l message_start is removed. This variable is only available if you have specified -l message_start.

Action variable	Description
\$year	The meaning of this variable is carrier library dependent. Not available on Palm.
\$month	The meaning of this variable is carrier library dependent. Values can be from 1-12. Not available on Palm.
\$day	The meaning of this variable is carrier library dependent. Values can be from 1-31. Not available on Palm.
\$hour	The meaning of this variable is carrier library dependent. Values can be from 0-23. Not available on Palm.
\$minute	The meaning of this variable is carrier library dependent. Values can be from 0-59. Not available on Palm.
\$second	The meaning of this variable is carrier library dependent. Values can be from 0-59. Not available on Palm.
\$type	The meaning of this variable is carrier library dependent. Not available on Palm.
\$priority	The meaning of this variable is carrier library dependent. Not available on Palm.
\$time	Palm only. This is the current time in seconds since 12:00 AM, January 1, 1904.

Default parameters file dblsn.txt

You can also create a file called dblsn.txt and specify arguments for dblsn in it. If you type dblsn without any arguments, dblsn will use dblsn.txt as the default argument file. This feature is particularly useful for CE devices.

On CE devices, dblsn.txt must be in the root directory. On Windows PCs, dblsn must be in the system path.

Palm devices

If you are using Palm remote devices, you must run the Listener utility (dbsln) on a Windows device to create a configuration file for the Palm. Use the dblsn -n option to create the configuration file. The configuration file must later be transferred to the Palm device via HotSync.

Once you have a configuration file on your Palm, you can use the **Palm Listener** utility to edit the message handlers in the configuration file. If you create a new configuration file, it will overwrite the old one.

The Palm Listener also allows you to set three semi-persistent options:

- ◆ **Listening** A way to stop the Listener from consuming messages. For example, if you turn off listening on a Treo 180, all messages will go into your default SMS message box on the device.
- ◆ **Enable Actions** This is applicable only when Listening is on.
- ◆ **Confirm Actions** This is applicable only when actions are enabled.

There is a separate Palm Listener for the two supported devices:

- ◆ For Kyocera 6035, use *lsnk6035.prc*
- ◆ For Treo 180, use *lsnT180.prc*

Notes

- ◆ When running the Listener on Windows to generate a configuration file for the Palm, you must specify an action. However, on the Palm device you can delete the action using the Handler Editor in the Palm Listener. This way you can consume the message without causing an action.
- ◆ The device need not be on if it turns on automatically when an SMS message is received. Kyocera and Treo devices do not need to be on for the Listener to work. The battery requirements of these devices are smaller.

Listener Software Development Kit

If you want to use remote devices that are not currently supported by MobiLink server-initiated synchronization, you can use the Listener Software Development Kit to create Listeners for those devices. The Listener SDK is a simple program API that is provided to help you extend the Listener utility.

You can use the Listener SDK to create Listeners for new Palm devices, or for new wireless network adapters for CE or laptops. The SDK provides development material for both Windows (32-bit and CE) and Palm operating systems.

The MobiLink Listener SDK and sample implementations are located in the following files. All are located in the *MobiLink\ListenerSDK* directory in your installation path.

Windows Files	Description
Win32andCE Win32_-VC lsn.def	Visual C++ module definition for the Listener library.
Win32andCE CE_EVC lsn.-def	Embedded Visual C module definition for the Listener library.
Win32andCE src lsn.h	Win32 and CE Listener library API.
Win32andCE src swi510.c	Sierra Wireless AirCard 510 implementation.
Win32andCE src udp.c	UDP implementation.
Palm Files	Description
Palm 68k cw lib PalmLsn.-lib	Runtime library for Palm Listeners. This provides a message handling routine, Listener controls, and a handler editor.
Palm 68k cw rsc . . .	Contains UI resources for the Palm Listener.
Palm src PalmLsn.h	Runtime library header and Palm Listener API.
Palm src Kyocera6035.c	Kyocera 6035 implementation.
Palm src Treo180.c	Treo180 implementation.

Deployment considerations

Following are some issues that you should consider before deploying server-initiated synchronization applications.

Limitations of UDP Listeners

- ◆ The UDP Listener keeps a socket open for listening, and so must be connected to an IP network all the time.
- ◆ The Listener does not detect lost connections and cannot re-open the listening socket.
- ◆ If you use dynamic IP addresses on the remote, you may have trouble updating the address on the server. MobiLink does not provide IP tracking. You may be able to use a third-party solution.
- ◆ The IP address on the remote needs to be reachable from the MobiLink synchronization server.

Limitations of Listeners on CE or PCs

- ◆ The Listener requires that the operating system is running, which could result in battery drain. Make sure that you have enough power for your usage pattern.

Delivery not guaranteed

- ◆ Both SMS and UDP delivery are not guaranteed, so you should implement a feedback loop if guaranteed delivery is essential. For example, you can use the MobiLink `last_upload_time` and `last_download_time` values to verify the delivery status of push requests.

Walkthrough of server-initiated synchronization

This section describes a hypothetical set of operations. It illustrates how server-initiated synchronization helps remote and consolidated databases stay in sync.

The walkthrough describes a situation with two remote databases (A and B). The following configuration parameters are set at the consolidated database:

- ◆ the Notifier polling interval is 2 minutes
- ◆ each push request has a resend interval of 5 minutes
- ◆ each push request has a time-to-live period of 6 minutes

The walkthrough starts just before noon on a business day. Initially, the remote databases A and B are both fully synchronized with the consolidated database. Here is a possible sequence of events.

1. On Remote A the Listener is on, but on Remote B the Listener is off.
2. Data is changed on the consolidated database.
3. The Notifier polls at 12:00:
 - ◆ The `begin_poll` statement is executed, and inserts push requests for A and B into the `PushRequest` table.
 - ◆ The `request_cursor SELECT` statement is executed to query the `PushRequest` table.
 - ◆ The Notifier sends messages to both remotes.
4. The Listener on Remote A picks up the message and invokes synchronization.

The synchronization resets the status of the push request and updates the `last_sync_time` for Remote A.
5. Being offline, Remote B does not receive the message and is not synchronized.
6. The next Notifier poll is two minutes later, at 12:02:
 - ◆ The `begin_poll` statement is executed. This won't insert a request for Remote A because its `last_sync_time` is greater than the `last_modified` time of the Dealer data. This won't insert a request for Remote B because the request has already been sent.
 - ◆ The `request_cursor` statement is executed to query the `PushRequest` table. The handled request for Remote A is not in the result set because the `last_sync_time` is greater than the `req_time` (but it is still in the `PushRequest` table). The request for Remote B remains in the result set.

- ◆ The `request_delete` statement is executed. This performs automatic cleanup of the request for Remote A because it is implicitly dropped.
 - ◆ The Notifier does not send a message to Remote B because the resend time is 12:05. The request for Remote B is pending for resend.
7. The next Notifier poll is at 12:04:
- ◆ The `begin_poll` statement is executed. This won't insert a request for Remote A because its `last_sync_time` is greater than the `last_modified` time of the Dealer data. This won't insert a request for Remote B because the request has already been inserted.
 - ◆ The `request_cursor` statement is executed to query the Request table. The request for Remote B remains in the result set.
 - ◆ The Notifier does not send a message to Remote B because the resend time is 12:05. The request for Remote B is pending for resend.
8. The next Notifier poll is at 12:05:
- ◆ The `begin_poll` statement is executed. This won't insert a request for Remote A because its `last_sync_time` is greater than the `last_modified` time of the Dealer data. This won't insert a request for Remote B because the request has already been inserted.
 - ◆ The `request_cursor` statement is executed to query the Request table. The request for Remote B remains in the result set.
 - ◆ The Notifier sends the message to Remote B again.
9. The next Notifier poll is at 12:07:
- ◆ The `begin_poll` statement is executed. This won't insert a request for Remote A because its `last_sync_time` is greater than the `last_modified` time of the Dealer data. This won't insert a request for Remote B because the request has already been inserted.
 - ◆ The `request_cursor` statement is executed to query the Request table. The request for Remote B remains in the result set.
 - ◆ The `request_delete` statement is executed. This performs automatic cleanup of the request for Remote B because it has exceeded its time to live.
 - ◆ There are no pending requests.
10. The next Notifier poll is at 12:09...

Sample applications

Several sample implementations of server-initiated synchronization are included in the SQL Anywhere Studio install. They are fully documented in readmes and code comments.


To locate the sample applications, navigate to the *Samples\MobiLink* directory in your SQL Anywhere Studio install path. All server-initiated synchronization sample directories start with the prefix *SIS_*.

CHAPTER 8

Adaptive Server Anywhere Clients

About this chapter

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

 For a tutorial to walk you through some of the concepts in this chapter, see [“Tutorial: Synchronizing Adaptive Server Anywhere Databases”](#) on page 369.




Contents

Topic:	page
Creating a remote database	168
Publishing data	171
Creating MobiLink users	178
Subscribing MobiLink synchronization users	182
Initiating synchronization	185
Using ActiveSync synchronization	189
Temporarily stopping synchronization of deletes	193
Customizing the client synchronization process	194
Scheduling synchronization	198
Adaptive Server Anywhere version 7 MobiLink clients	200

Creating a remote database

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

❖ To create an Adaptive Server Anywhere remote database

1. Start with an existing Adaptive Server Anywhere database, or create a new one and add your tables.
 See [“Publishing data” on page 171](#).
2. Create one or more publications in the new database.
 See [“Creating MobiLink users” on page 178](#).
3. Create a MobiLink user.
 See [“Subscribing MobiLink synchronization users” on page 182](#).
4. Subscribe a MobiLink user to one or more of the publications.

Deploying remote databases

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

❖ To deploy MobiLink remote databases by customizing a prototype

1. Create a prototype remote database.
The prototype database should have all the tables and publications needed, but not the information that is specific to each database. This individual information typically includes the following:
 - ◆ The MobiLink user name.
 - ◆ Synchronization subscriptions.
 - ◆ The GLOBAL_DATABASE_ID option that provides the starting point for global autoincrement key values.
2. For each remote database, carry out the following operations:
 - ◆ Create a directory to hold the remote database.
 - ◆ Copy the prototype remote database into the directory.
If the transaction log is held in the same directory as the remote database, the log filename does not need to be changed.

- ◆ Run a SQL script that adds the individual information to the database. The SQL script can be a parameterized script. For information on parameterized scripts, see “PARAMETERS statement [Interactive SQL]” [ASA SQL Reference, page 506], and “Running SQL command files” [ASA SQL User's Guide, page 553].

Example

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

```
PARAMETERS ml_userid, db_id;
go
SET OPTION PUBLIC.GLOBAL_DATABASE_ID = {db_id}
go

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

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

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

Partitioning data between remote databases

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

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

You can further fine-tune the data any given remote database will receive by using a WHERE clause within the publication. This feature is useful when remote databases require similar types of information. For example, it can

be used to provide sales representatives with only the information relevant to their region.

☞ For more information, see [“Partitioning rows among remote databases” on page 77](#).

Upgrading remote databases

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

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

☞ For more information, see “Upgrading Adaptive Server Anywhere MobiLink clients” [*What’s New in SQL Anywhere Studio*, page 185].

Publishing data

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

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

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

- Notes about publications
- ◆ DBA authority is required to create and drop publications.
 - ◆ A single publication can publish a subset of columns from a set of tables and use a `WHERE` clause to select a set of rows to be replicated.
 - ◆ Views and stored procedures cannot be included in publications.
 - ◆ Publications and subscriptions are also used by the Sybase message-based replication technology, SQL Remote. SQL Remote requires publications and subscriptions in both the consolidated and remote databases. In contrast, MobiLink publications appear only in Adaptive Server Anywhere remote databases. MobiLink consolidated databases are configured using synchronization scripts.

Publishing whole tables

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

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

1. Connect to the remote database as a user with DBA authority, using the Adaptive Server Anywhere plug-in.
2. Open the Publications folder.
3. From the File menu, choose **New ► Publication**. The Create a New Publication wizard appears.
4. Type a name for the new publication. Click **Next**.
5. On the **Tables** tab, select a table from the list of Available Tables. Click **Add**. The table appears in the list of Selected Tables on the right.

-
6. Optionally, you may add additional tables. The order of the tables is not important.
 7. Click Finish.

❖ **To publish one or more entire tables (SQL)**

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

Example

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

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

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

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

☞ For more information, see the “CREATE PUBLICATION statement” [ASA SQL Reference, page 334].

Publishing only some columns in a table

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

Note

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

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

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

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

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

Example

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

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

☞ For more information, see the “CREATE PUBLICATION statement” [ASA SQL Reference, page 334].

Publishing only some rows in a table

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

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

◆ **WHERE clause** Compatible with MobiLink. You can use a WHERE clause to include a subset of rows in an article.

◆ **Subscription expression** Ignored by MobiLink.

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

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

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

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

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

Examples

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

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

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

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

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

☞ For more information, see the “CREATE PUBLICATION statement” [ASA *SQL Reference*, page 334]. Note that the CREATE PUBLICATION statement includes a SUBSCRIBE BY clause. This clause can be used to selectively publish rows in SQL Remote. However, it is ignored during MobiLink synchronization.

Altering existing publications

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

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

Notes

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

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

1. Connect to the remote database as a user who owns the publication or as a user with DBA authority.
2. In the left pane, click the publication or article. The properties will appear in the right pane.
3. Configure the desired properties.

❖ To add articles (Sybase Central)

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

❖ To remove articles (Sybase Central)

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

❖ To modify an existing publication (SQL)

1. Connect to the remote database as a user who owns the publication or as a user with DBA authority.
 2. Connect to a database with DBA authority.
 3. Execute an ALTER PUBLICATION statement.
- ◆ The following statement adds the customer table to the pub_contact publication.

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

Example

☞ See also the “ALTER PUBLICATION statement” [ASA *SQL Reference*, page 238].

Dropping publications

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

You must have DBA authority to drop a publication.

❖ To delete a publication (Sybase Central)

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

❖ To delete a publication (SQL)

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

Example

The following statement drops the publication named pub_orders.

```
DROP PUBLICATION pub_orders
```

☞ See also the “DROP PUBLICATION statement” [ASA *SQL Reference*, page 413].

Creating MobiLink users

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

MobiLink users are not the same as database users. You can create a MobiLink user ID that matches the name of a database user, but neither MobiLink nor Adaptive Server Anywhere is affected by this coincidence.

☞ For information about adding MobiLink users to the consolidated database, see [“About MobiLink users” on page 104](#).

Adding MobiLink users to a remote database

This section describes how to add a MobiLink user name to a remote database. For information on supplying MobiLink user properties, including the password, see [“Configuring MobiLink user properties” on page 179](#).

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

1. Connect to the database from the Adaptive Server Anywhere plug-in as a user with DBA authority.
2. Click the MobiLink Users folder.
3. From the File menu, choose New ► MobiLink User. The Create a New MobiLink User wizard appears.
4. Enter a name for the MobiLink user. This name is supplied to the MobiLink synchronization server during synchronization.
5. Click Finish.

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

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

The following example adds a MobiLink user named SSinger:

```
CREATE SYNCHRONIZATION USER SSinger
```

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

☞ For more information, see [“CREATE SYNCHRONIZATION USER statement \[MobiLink\]” \[ASA SQL Reference, page 351\]](#).

Configuring MobiLink user properties

You can specify the following properties for each MobiLink user in a remote database:

- ◆ **Connection properties** This information includes the address for the MobiLink synchronization server, the protocol to use for communications with the server, and other connection parameters.

☞ For more information, see “CREATE SYNCHRONIZATION USER statement [MobiLink]” [*ASA SQL Reference*, page 351].

- ◆ **Extended options** Extended options include the password (although it is more secure to supply a password at synchronization time, and use it on the dbmlsync command line), the script version, as well as options that tune performance and behavior.

☞ For more information, see “-e extended options” [*MobiLink Synchronization Reference*, page 44].

❖ To configure MobiLink user properties (Sybase Central)

1. Connect to the database from the Adaptive Server Anywhere plug-in as a user with DBA authority.
2. Open the MobiLink Users folder.
3. Right-click the MobiLink user name and choose Properties from the pop-up menu.
4. Change the properties as needed.

❖ To configure MobiLink user properties (SQL)

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

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

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

☞ For more information, see “ALTER SYNCHRONIZATION USER statement [MobiLink]” [*MobiLink Synchronization Reference*, page 238].

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

☞ For more information, see “CREATE SYNCHRONIZATION USER statement [MobiLink]” [*MobiLink Synchronization Reference*, page 245].

Priority order for extended options and connection parameters

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

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

If extended options are specified in both the database and the command line, the option strings are combined. If conflicting options are specified, dbmlsync resolves them as follows. In the following list, options specified by methods occurring earlier in the list take precedence over those occurring later in the list.

1. options specified on the command line with the dbmlsync -eu option.
2. options specified on the command line with the dbmlsync -e option.
3. options specified for the subscription (whether using SQL statements or Sybase Central).
4. options specified for the user (whether using SQL statements or Sybase Central).
5. options specified for the publication (whether using SQL statements or Sybase Central).

If the connection TYPE or ADDRESS is specified in more than one place, the one specified with the highest priority according to the list above overrides any other specification. The connection TYPE and ADDRESS can be specified on the command line using adr and ctp extended options.

Dropping MobiLink users

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

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

1. Connect to the database from the Adaptive Server Anywhere plug-in as a user with DBA authority.
2. Locate the MobiLink user in the MobiLink Users folder.
3. Right click the MobiLink user and choose Delete from the popup menu.

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

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

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

```
DROP SYNCHRONIZATION USER SSinger
```

☞ For more information, see “DROP SYNCHRONIZATION USER statement [MobiLink]” [*MobiLink Synchronization Reference*, page 257].

Subscribing MobiLink synchronization users

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

☞ For information about creating publications, see [“Publishing data” on page 171](#). For information about creating MobiLink users, see [“Creating MobiLink users” on page 178](#).

Subscriptions versus synchronization subscriptions

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

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

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

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

Example

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

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

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

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

```
CREATE SYNCHRONIZATION USER demo_ml_user
```

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

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

Altering MobiLink subscriptions

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

❖ To alter a synchronization subscription (Sybase Central)

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

❖ To alter a synchronization subscription (SQL)

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

☞ For more information, see “`ALTER SYNCHRONIZATION SUBSCRIPTION` statement [MobiLink]” [*MobiLink Synchronization Reference*, page 236].

Dropping MobiLink subscriptions

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

You must have DBA authority to drop a synchronization subscription.

❖ **To delete a synchronization subscription (Sybase Central)**

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

❖ **To delete a synchronization subscription (SQL)**

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

Example

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

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

☞ See also the “DROP SYNCHRONIZATION SUBSCRIPTION statement [MobiLink]” [ASA *SQL Reference*, page 420].

Initiating synchronization

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

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

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

Permissions for `dbmlsync`

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

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

Example

Suppose that you have a remote database named `remote` and that this database is currently running on your local machine. In addition, assume that the MobiLink synchronization server has been started and is ready to accept requests. You could use the following command to synchronize as user `syncuser`, who has been granted REMOTE DBA authority.

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

Since the user's password is not specified on the command line, a dialog appears letting you enter this additional piece of information.

Note that no connection parameters for the MobiLink synchronization server appear on the command line. Instead, these parameters are set in the synchronization subscription, publication, or user, and stored in the remote database.

Multiple MobiLink synchronization users

Each remote database typically contains exactly one MobiLink synchronization user. In this case, you do not need to specify a MobiLink user name on the `dbmlsync` command line. However, if the remote database

contains more than one, you must specify which MobiLink synchronization user to synchronize using the `-u` command line option.

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

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

Customizing synchronization

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

☞ For a complete list of extended options, see “-e extended options” [*MobiLink Synchronization Reference*, page 44].

❖ To override an extended option on the `dbmlsync` command line

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

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

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

1. Add the option to the `CREATE SYNCHRONIZATION SUBSCRIPTION` statement or `CREATE SYNCHRONIZATION USER` statement in the Adaptive Server Anywhere remote database.

Adding an extended option for a publication is a little different. To add an extended option for a publication, use the `ALTER/CREATE SYNCHRONIZATION SUBSCRIPTION` statement and omit the `FOR` clause.

Example

The following statement creates a synchronization subscription that uses extended options to set the cache size for preparing the upload stream to 3 Mb and the upload increment size to 3 kb.

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

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

Transaction log files

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

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

☞ For more information, see “Backup and Data Recovery” [ASA Database Administration Guide, page 337].

Example

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

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

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

Concurrency during synchronization

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

By default, dbmlsync obtains an exclusive lock on all tables mentioned in any publication being synchronized. It does this before it begins building the upload stream. Dbmlsync maintains this lock until the download is applied.

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

☞ For more information, see “-d option” [*MobiLink Synchronization Reference*, page 42].

You can further protect data integrity by setting the extended option `LockTables` to `OFF`. This causes `dbmsync` to track all rows that are modified after the upload stream has been built. When the download is received, it is not applied if any rows in the download have been modified. `Dbmsync` will then retry the synchronization. The retry will succeed unless a new download conflict is detected.

☞ For more information, see “`LockTables (lt)` extended option” [*MobiLink Synchronization Reference*, page 55].

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

☞ For more information, see “`ConflictRetries (cr)` extended option” [*MobiLink Synchronization Reference*, page 47].

Initiating synchronization from an application

You may wish to include the features of `dbmsync` in your application, rather than provide a separate executable to your customers. If you are developing in any language that can call a DLL, you can do so.

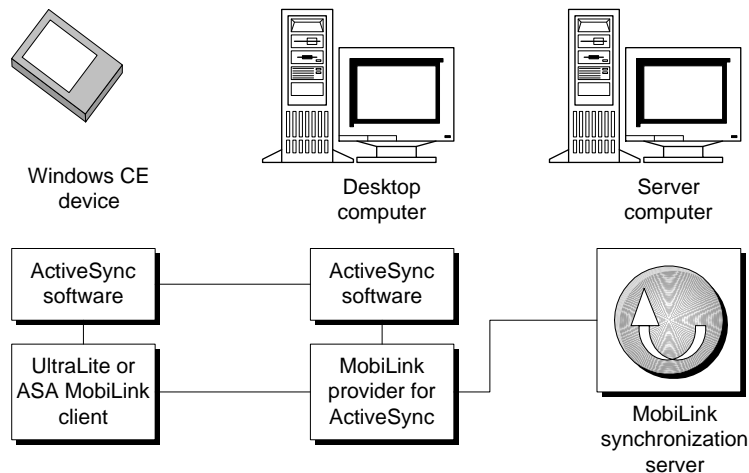
If you are programming in C or C#, include the `dbtools.h` header file located in the `h` subdirectory of your SQL Anywhere directory. This file contains a description of the `a_sync_db` structure and the `DBSynchronizeLog` function, which you use to add this functionality to your application.

☞ For more information, see “`DBSynchronizeLog` function” [*ASA Programming Guide*, page 273], and “`a_sync_db` structure” [*ASA Programming Guide*, page 295].

Using ActiveSync synchronization

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

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



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

- ◆ Configure the Adaptive Server Anywhere remote database for ActiveSync synchronization.
 - ☞ See [“Configuring Adaptive Server Anywhere remote databases for ActiveSync” on page 190.](#)
- ◆ Install the MobiLink provider for ActiveSync.
 - ☞ See [“Installing the MobiLink provider for ActiveSync” on page 191.](#)
- ◆ Register the Adaptive Server Anywhere client for use with ActiveSync.
 - ☞ See [“Registering Adaptive Server Anywhere clients for ActiveSync” on page 192.](#)

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

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

☞ For information about scheduling synchronization, see [“Scheduling synchronization” on page 198](#).

Configuring Adaptive Server Anywhere remote databases for ActiveSync

❖ To configure your Adaptive Server Anywhere remote database for ActiveSync

1. Select ActiveSync as the synchronization type.

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

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

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

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

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

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

☞ For more information, see “CREATE SYNCHRONIZATION USER statement [MobiLink]” [ASA SQL Reference, page 351].

Installing the MobiLink provider for ActiveSync

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

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

When you have installed the MobiLink provider for ActiveSync you must register each application separately. For instructions, see [“Registering Adaptive Server Anywhere clients for ActiveSync” on page 192](#).

❖ To install the MobiLink provider for ActiveSync

1. Ensure that you have the ActiveSync software on your machine, and that the Windows CE device is connected.
2. Enter the following command to install the MobiLink provider:

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

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

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

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

☞ For more information, see “ActiveSync provider installation utility” [*MobiLink Synchronization Reference*, page 300].

3. Restart your machine.

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

4. Enable the MobiLink provider.

- ◆ From the ActiveSync window, click Options.
- ◆ Check the MobiLink item in the list and click OK to activate the provider.

-
- ◆ To see a list of registered applications, click Options again, choose the MobiLink provider, and click Settings.
 - ☞ For more information about registering applications, see [“Registering Adaptive Server Anywhere clients for ActiveSync” on page 192.](#)

Registering Adaptive Server Anywhere clients for ActiveSync

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

☞ For information on the alternative approach, see “ActiveSync provider installation utility” [*MobiLink Synchronization Reference*, page 300].

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

1. Ensure that the MobiLink provider for ActiveSync is installed.
 - ☞ For information, see [“Installing the MobiLink provider for ActiveSync” on page 191.](#)
2. Start the ActiveSync software on your desktop machine.
3. From the ActiveSync window, choose Options.
4. From the list of information types, choose MobiLink and click Settings.
5. In the MobiLink Synchronization dialog, click New. The Properties dialog appears.
6. Enter the following information for your application:
 - ◆ **Application name** A name identifying the application to be displayed in the ActiveSync user interface.
 - ◆ **Class name** The class name for the dbmlsync client, as set using its `-wc` option.
 - ☞ For more information, see “MobiLink synchronization client” [*MobiLink Synchronization Reference*, page 36].
 - ◆ **Path** The location of the dbmlsync application on the device.
 - ◆ **Arguments** Any command line arguments to be used when ActiveSync starts dbmlsync.
7. Click OK to register the application.

Temporarily stopping synchronization of deletes

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

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

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

❖ To temporarily disable upload of deletes made through a connection

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

```
STOP SYNCHRONIZATION DELETE
```

2. Delete rows from the synchronized data, as required, using the `DELETE` statement. Commit these changes.
3. Restart logging of deletes using the following statement.

```
START SYNCHRONIZATION DELETE
```

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

Customizing the client synchronization process

The Adaptive Server Anywhere synchronization client, dbmlsync, provides a set of event hooks that you can use to customize the synchronization process. When a hook is implemented, it is called at a specific time during synchronization. You implement an event hook by creating a stored procedure with a specific name. Most event-hook stored procedures are executed on the same connection as the synchronization itself.

You can use event hooks to delay synchronization until a specific condition is met, such as the total number of changes made reaches a set number, a particular change is made, or some data-independent condition.

In addition, you can use event hooks to synchronize subsets of data that cannot be easily defined in a publication. For example, you can synchronize data in a temporary table by writing one event hook procedure to copy data from the temporary table to a permanent table prior to the synchronization and another to copy the data back afterwards.

☞ For more information about specific event-hook procedures, see “Client event-hook procedures” [*MobiLink Synchronization Reference*, page 269].

Caution

The integrity of the synchronization process relies on a sequence of built-in transactions. Thus, you must not perform an implicit or explicit commit or rollback within your event-hook procedures.

Synchronization event hook sequence

The following pseudo-code shows the available events and the point at which each is called during the synchronization process. For example, `sp_hook_dbmlsync_abort` is the first event hook to be invoked.

Each event makes particular parameter values available, which you can use when you implement the procedure. In some cases, you can modify the value to return a new value; others are read-only. These parameters are not stored procedure arguments. No arguments are passed to any of the event-hook stored procedures. Instead, arguments are exchanged by reading and modifying rows in the `#hook_dict` table.

For example, the `sp_hook_dbmlsync_begin` procedure has a parameter, which is the user name that the application supplied in the synchronization call. You can retrieve this value from the `#hook_dict` table.

Although the sequence has similarities to the event sequence at the MobiLink synchronization server, there is little overlap in the kind of logic

you would want to add to the consolidated and remote databases. The two interfaces are therefore separate and distinct.

Any *_end hook will be called if the corresponding _begin hook is called and completed successfully. A *_begin hook is considered to have run successfully if it was not implemented when it would have been called.

```

sp_hook_dbmlsync_abort
loop until return codes direct otherwise (
    sp_hook_dbmlsync_abort
    sp_hook_dbmlsync_delay
)
sp_hook_dbmlsync_abort
// start synchronization
sp_hook_dbmlsync_begin
// upload events
sp_hook_dbmlsync_logscan_begin
sp_hook_dbmlsync_logscan_end
sp_hook_dbmlsync_upload_begin
sp_hook_dbmlsync_upload_end
// download events
sp_hook_dbmlsync_validate_download_file (only called
    when -ba option is used)
sp_hook_dbmlsync_download_begin
for each table
    sp_hook_dbmlsync_download_table_begin
    sp_hook_dbmlsync_download_table_end
next table
sp_hook_dbmlsync_download_end
// end synchronization
sp_hook_dbmlsync_end
sp_hook_dbmlsync_process_return_code

```

Error handling

In addition, the following event-hook procedures are available for error handling.

```

sp_hook_dbmlsync_download_com_error
sp_hook_dbmlsync_download_SQL_error
sp_hook_dbmlsync_download_fatal_SQL_error
sp_hook_dbmlsync_download_ri_violation
sp_hook_dbmlsync_download_log_ri_violation

```

Once implemented, each procedure is automatically executed whenever an error of the named type occurs.

☞ For more information, see “Client event-hook procedures” [*MobiLink Synchronization Reference*, page 269].

Using event-hook procedures

This section describes some considerations for designing and using event-hook procedures.

Event-hook procedure owner

The event-hook connection calls the stored procedures without qualifying them by owner. The stored procedures must therefore be owned by one of the following:

- ◆ The user name employed on the dbmsync connection (typically a user with REMOTE DBA authority).
- ◆ A group ID of which the dbmsync user is a member.

Connections for event-hook procedures

Each event-hook procedure is executed on the same connection as the synchronization itself. The following are exceptions:

- ◆ `sp_hook_dbmsync_download_com_error`
- ◆ `sp_hook_dbmsync_download_fatal_sql_error`
- ◆ `sp_hook_dbmsync_download_log_ri_violation`

These procedures are called before a synchronization fails. On failure, synchronization actions are rolled back. By operating on a separate connection, you can use these procedures to log information about the failure, without the logging actions being rolled back along with the synchronization actions.

Event arguments

Each hook receives parameter values. In some cases, you can modify the value to return a new value; others are read-only.

These parameters are exchanged by reading and modifying rows in the `#hook_dict` table, which is defined as follows.

```
CREATE TABLE #hook_dict (
    name    VARCHAR( 128 ) NOT NULL UNIQUE,
    value   VARCHAR( 255 ) NOT NULL
)
```

Each row in the table contains the value for one parameter.

Before calling any of the stored procedures, dbmsync creates the `#hook_dict` table, and adds the parameters for that event. Procedures can read the values by selecting from this table.

Some parameters can be used to pass values back to dbmsync from the hook. The hook passes values back by updating the `#hook_dict` table.

☞ For a list of the parameter values supplied at each event, see “Client event-hook procedures” [*MobiLink Synchronization Reference*, page 269].

Examples

The following examples illustrate how to retrieve and set values in the #hook_dict table.

The following sample sp_hook_dbmlsync_delay procedure illustrates the use of the #hook_dict table to pass arguments. The procedure allows synchronization only outside a scheduled down time of the MobiLink system between 18:00 and 19:00.

```
CREATE PROCEDURE sp_hook_dbmlsync_delay()
BEGIN
    DECLARE delay_val integer;
    SET delay_val=DATEDIFF(
        second, CURRENT TIME, '19:00');
    IF (delay_val>0 AND
        delay_val<3600)
    THEN
        UPDATE #hook_dict SET value=delay_val
            WHERE name='delay duration';
    END IF;
END
```

The following procedure is executed in the remote database at the beginning of synchronization. It retrieves the current MobiLink user name, one of the parameters available for the sp_hook_dbmlsync_begin event, and displays it on the console.

```
CREATE PROCEDURE sp_hook_dbmlsync_begin()
BEGIN
    DECLARE syncdef VARCHAR(128);
    SELECT '>>>syncdef = ' || value INTO syncdef
        FROM #hook_dict
        WHERE name = 'MobiLink user name';
    MESSAGE syncdef TYPE INFO TO CONSOLE;
END
```

Ignoring errors in event-hook procedures

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

Scheduling synchronization

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

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

☞ For information about extended options, see “-e extended options” [*MobiLink Synchronization Reference*, page 44] or “-eu option” [*MobiLink Synchronization Reference*, page 71]. For more information about how to set scheduling, see “Schedule (sch) extended option” [*MobiLink Synchronization Reference*, page 59].

❖ To add scheduling to the synchronization subscription

1. Set the Schedule extended option in the synchronization subscription. For example,

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

☞ For more information about scheduling syntax, see “Schedule (sch) extended option” [*MobiLink Synchronization Reference*, page 59].

☞ You can override scheduling instructions and synchronize immediately using the `dbmlsync -is` option. The `-is` option instructs `dbmlsync` to ignore all scheduling information. For more information, see “-is option” [*MobiLink Synchronization Reference*, page 72].

❖ To add scheduling from the dbmlsync command line

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

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

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

Hovering

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

☞ For more information, see “HoverRescanThreshold (hrt) extended option” [*MobiLink Synchronization Reference*, page 52] and “sp_hook_dbmlsync_log_rescan” [*MobiLink Synchronization Reference*, page 286].

Adaptive Server Anywhere version 7 MobiLink clients

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

1. A synchronization definition is equivalent to a single publication and a single subscription to it. There is no support for subscriptions to multiple publications. In contrast, a single MobiLink user can now subscribe to multiple publications. This allows you to synchronize some portions of your data without synchronizing all of it.
2. Some people found the old terminology confusing. For example, a MobiLink user ID was formerly called a site in the context of an Adaptive Server Anywhere client. A MobiLink user is now called a MobiLink user or a synchronization user.
3. The new statements are analogous to those used in SQL Remote, the Sybase message-based replication technology.

Synchronization definitions identify data to upload in version 7 remote databases

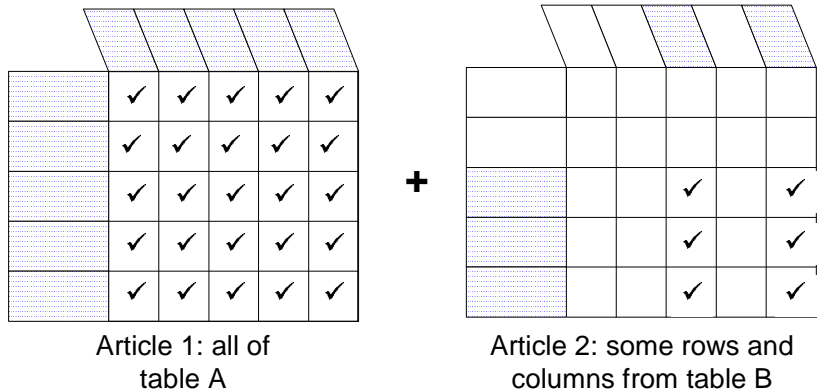
You can choose to synchronize all or any portion of the data in a client Adaptive Server Anywhere database. You can choose to synchronize entire tables, or you can choose to synchronize only particular columns and rows.

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

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

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

A two-table synchronization definition



Synchronizing a remote database

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

Altering a synchronized table

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

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

Comparison to UltraLite clients

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

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

Writing synchronization definitions

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

☞ For details, see [“Creating a remote database” on page 168](#).

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

A synchronization definition specifies the following pieces of information

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

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

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

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

In this statement,

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

The TCP/IP connection parameters show that the MobiLink synchronization server is listening on port 2439 of the current machine. Only the listed columns of the `People` table are synchronized. The option clause is included to indicate that the MobiLink synchronization server should use *version1* of the synchronization scripts when processing data from this client. The default value of this parameter is *default*. Notice that the list of columns is also enclosed in parentheses.

- ◆ The MobiLink synchronization server is to use the set of synchronization scripts identified by the name `version1` when synchronizing this client. This script version name should appear in the `ml_script_version` MobiLink system table, located in the consolidated database.
- ◆ All columns and rows of the `Pets` table and the listed columns of the `People` table are to be synchronized.

☞ For the syntax of the MobiLink-synchronization-specific statements, see “SQL Statements” [*MobiLink Synchronization Reference*, page 233].

Synchronizing with
multiple servers

To synchronize a remote database with multiple MobiLink synchronization servers, create multiple synchronization definitions within the remote database. Each synchronization definition must have a unique site name

Rewriting
synchronization
definitions for version 8
and up

Example

because, from the point of view of the MobiLink synchronization server, each is a separate logical client.

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

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

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

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

Instead, you should now do the following.

1. First, publish the Customer and Sales_Order tables.

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

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

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

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

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

In the second case, the synchronized tables are published, and then a subscription is created for the demo_sync_site MobiLink user. The TYPE,

ADDRESS, and OPTION clauses have the same syntax.

CHAPTER 9

UltraLite Clients

About this chapter

This chapter describes how to use an UltraLite database as a MobiLink client. It introduces synchronization streams and provides material on how to set up Palm OS devices and Windows CE devices for synchronization.

☞ For more information about UltraLite remote databases and MobiLink, see “Synchronization for UltraLite Applications” [*UltraLite Database User’s Guide*, page 143].

Contents

Topic:	page
Introduction to synchronization streams	208
Synchronizing UltraLite databases on the Palm Computing Platform	209
Synchronizing UltraLite databases on Windows CE	223

Introduction to synchronization streams

Each UltraLite database that synchronizes with a MobiLink synchronization server does so over a synchronization stream. The synchronization stream is specified in the UltraLite application. Available synchronization streams include TCP/IP, HTTP, and HTTPS for TCP/IP based networks. Support is also provided for HotSync synchronization on the Palm Computing Platform and for ActiveSync synchronization on Windows CE.

☞ For more information, see “Selecting a synchronization stream” [*UltraLite Database User’s Guide*, page 147].

Synchronizing UltraLite databases on the Palm Computing Platform

This section describes the details of synchronization that are specific to the Palm Computing Platform.

☞ For more information about UltraLite on the Palm Computing Platform, see “Developing UltraLite Applications for the Palm Computing Platform” [*UltraLite Embedded SQL User’s Guide*, page 71].

Choosing a synchronization method

Synchronization on the Palm Computing Platform can be carried out using HotSync or over standard network protocols using TCP/IP or HTTP. Each synchronization method has its advantages and disadvantages.

- ◆ **Multiple applications** If you have more than one UltraLite application installed on a Palm device, they all synchronize when you invoke HotSync. To synchronize multiple applications through a TCP/IP or HTTP connection, you must activate and synchronize each application in turn.
- ◆ **Universal Serial Bus support** HotSync synchronization has automatic support for USB.
- ◆ **Publications** Synchronization using HotSync cannot include WHERE clauses.

☞ For more information, see “Designing sets of data to synchronize separately” [*UltraLite Database User’s Guide*, page 156].

Understanding HotSync synchronization

UltraLite applications on Palm devices can synchronize over a TCP/IP or HTTP stream, in much the same manner as UltraLite applications on other platforms. They can also synchronize using the Palm-specific HotSync synchronization streams, which operate in a different manner. This section describes the architecture of the HotSync synchronization.

The sequence of events that occur during HotSync synchronization is as follows:

1. When your UltraLite application is closed, it saves the state of your UltraLite application. The state information is stored in the Palm database, separately from the UltraLite database.

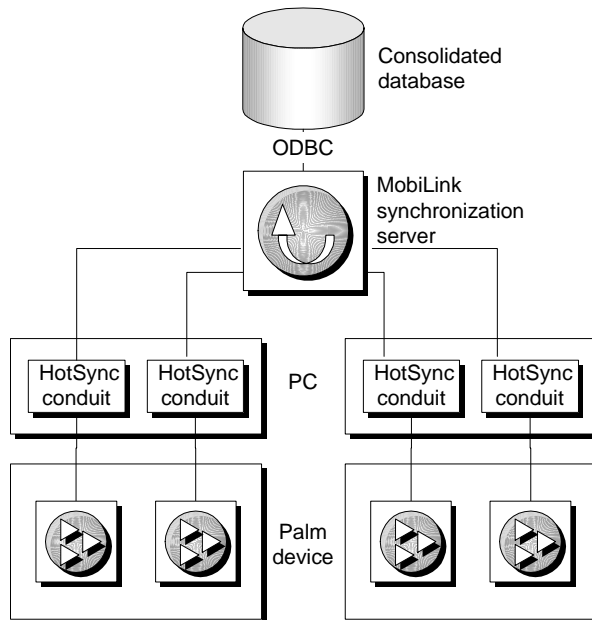
☞ For more information, see “Closing an UltraLite Palm application” [*UltraLite Embedded SQL User’s Guide*, page 77].

2. When you synchronize your Palm device, HotSync calls the MobiLink conduit to synchronize with the MobiLink synchronization server. The MobiLink conduit reads the pages from the UltraLite database and sends the upload to the MobiLink synchronization server.
3. The MobiLink synchronization server integrates updates into the consolidated database and sends a download stream to the conduit.
4. The conduit integrates the download stream into the UltraLite database on the Palm device.
5. When your application is launched, it loads the previously saved state of your UltraLite application.

☞ For more information, see “Launching an UltraLite Palm application” [*UltraLite Embedded SQL User’s Guide*, page 77].

HotSync architecture

The following diagram depicts the HotSync architecture. A separate HotSync conduit is required for each application. You can have multiple HotSync conduits on a single PC.



☞ For a description of how to set up your MobiLink HotSync conduit, see “[Configuring the MobiLink HotSync conduit](#)” on page 214.

HotSync configuration overview

During HotSync synchronization, the HotSync Manager starts the MobiLink HotSync conduit, *dbhsync9.dll*, which reads from the device and then sends the upload stream to a MobiLink synchronization server. It then receives the download stream from the MobiLink synchronization server and writes the download to the device.

The MobiLink HotSync conduit synchronizes with the MobiLink synchronization server using one of TCP/IP, HTTP, or HTTPS streams.

In most applications, only the MobiLink HotSync conduit is deployed onto the desktop machines of users.

☞ For information about HotSync architecture, see [“Understanding HotSync synchronization” on page 209](#).

❖ To install and configure the MobiLink HotSync conduit

1. Place the MobiLink conduit files on the user’s machine.
☞ For instructions, see [“Conduit files” on page 211](#).
2. Register the MobiLink conduit to the HotSync Manager. The HotSync Manager is then able to use the MobiLink conduit.
☞ For instructions, see [“Registering the MobiLink HotSync conduit to HotSync Manager” on page 212](#).
3. If you did not include a **stream_parms** parameter in your UltraLite **ul_synch_info** structure, enter these parameters from the HotSync Manager.
☞ For instructions, see [“Configuring the MobiLink HotSync conduit” on page 214](#).
☞ For information about including **stream_parms** parameter in your UltraLite synchronization call, see [“Adding HotSync synchronization to Palm applications”](#) [*UltraLite Embedded SQL User’s Guide*, page 81].
4. If you are using an encrypted database, enter the encryption key in the conduit configuration dialog. If you do not enter this key, you will have to enter it on every synchronization.
☞ For instructions, see [“Configuring the MobiLink HotSync conduit” on page 214](#).

Conduit files

The HotSync conduit consists of the following files:

- ◆ **dbhsync9.dll** The DLL that is called by the HotSync Manager.

-
- ◆ **dblggen9.dll** The language resource library. For languages other than English, the letters en in the file name are replaced by a two-letter abbreviation for the language, such as *dblgde9.dll* or *dblgja9.dll*.
 - ◆ **Stream DLL** You need a DLL for the communication between the conduit and the MobiLink synchronization server. A separate DLL is provided for each stream:
 - For TCP/IP, use *dbmlsock9.dll*.
 - For HTTP, use *dbmlsock9.dll* and *dbmlhttp9.dll*.
 - For HTTPS, use *dbmlhttps9.dll*
 - If you use encryption for this communication, you also need to supply the encryption DLL *dbmltls9.dll*.

These files should be in the same directory, in your system path. When you install SQL Anywhere Studio, they are installed into the operating system subdirectory of your installation directory, which is already in the system path. However, you do not have to install SQL Anywhere Studio to use these files.

Registering the MobiLink HotSync conduit to HotSync Manager

UltraLite includes a command line **conduit installation utility** named *dbcond9* to make a set of registry entries for the HotSync Manager to be able to use the MobiLink conduit. This utility requires the following files:

- ◆ *dbcond9.exe*
- ◆ *condmgr.dll*

❖ To deploy the conduit installation utility

1. Choose a top-level deployment directory.

For example, you may choose a directory named *c:\deploy*.

2. Add a registry entry with the full path of the deployment directory as its value.

The registry entry must be as follows:

```
HKEY_CURRENT_USER\Software\Sybase\Adaptive Server Anywhere\  
version-string\Location
```

where *version-string* is a number representing your version of the SQL Anywhere Studio (such as **9.0**). If the entry is not found in *HKEY_CURRENT_USER*, the software looks in *HKEY_LOCAL_MACHINE*.

3. Add the *dbcond9.exe* file to the *win32* subdirectory of the deployment directory.
4. Add the *condmgr.dll* file.

The *condmgr.dll* file must go in the *win32\condmgr* subdirectory of the deployment directory.

The SQL Anywhere Studio installation creates the required registry entries and places files in the appropriate locations.

❖ To register the MobiLink HotSync conduit to HotSync Manager

1. Ensure that the HotSync conduit files and the files for the conduit installation utility are in place.
2. Run the conduit installation utility. On the command line, you must specify the creator ID of the Palm application and a name that HotSync will use to identify the conduit.

For example, the following command installs a conduit for the application with creator ID **Syb2**, named **CustDB**. These are the settings for the CustDB sample application:

```
dbcond9 "Syb2" -n CustDB
```

☞ For more information about the conduit installation utility, see “The HotSync conduit installation utility” [*UltraLite Database User’s Guide*, page 91].

Note

A secondary location for HotSync synchronization depends on the version of the Palm Computing Platform software you are using. This secondary location may be under the *HKEY_CURRENT_USER\Software\U.S. Robotics* or *HKEY_CURRENT_USER\Software\Palm Computing* folders.

Checking that MobiLink HotSync conduit installation is correct

Following are instructions for verifying that your conduit is installed and is working.

❖ **To check that the HotSync conduit is properly installed**

1. Check that a conduit is installed:
 - ◆ In your PC's system tray, right-click HotSync Manager.
 - ◆ From the pop-up menu, choose Custom.
A list of conduits is displayed for each user. Verify that your conduit is listed.
2. Set the system environment variable `UL_DEBUG_CONDUIT` to any value.
3. Shut down and restart the HotSync Manager.
4. If the MobiLink conduit is properly installed, two dialog boxes appear. If no dialog appears, the conduit was not properly installed.
5. Unset the environment variable.
6. Shut down and restart the HotSync Manager.

MobiLink must be started before using HotSync

Before using HotSync, the MobiLink synchronization server must be started and be ready to accept connections from the MobiLink HotSync conduit. The MobiLink synchronization server does not have to be on the same computer, but it must be reachable across the network.

Configuring the MobiLink HotSync conduit

The MobiLink HotSync conduit needs to communicate with a MobiLink synchronization server to synchronize the UltraLite application and the consolidated database. You can provide the information needed by the conduit to locate the MobiLink synchronization server in a **stream_parms** member of the UltraLite **ul_synch_info** structure supplied to the **PalmExit** function. If you did not specify a **stream_parms** value, or if you specified the value as null, you can enter the required parameters from the HotSync Manager.

In addition, if you are using a strongly encrypted UltraLite database, you can save the encryption key so that you do not have to enter it on each synchronization.

If you have Palm Desktop software installed, the Adaptive Server Anywhere installation creates registry entries for the **CustDB** sample application. You can use these entries as a starting point for your own application.

☞ For information about **stream_parms**, see “Adding HotSync synchronization to Palm applications” [*UltraLite Embedded SQL User’s Guide*, page 81].

❖ To configure the MobiLink HotSync conduit for synchronization

1. Right-click the HotSync Manager icon in the system tray, and choose Custom from the popup menu.
2. Select your MobiLink HotSync conduit from the list of conduit names, and click Change.
3. Enter a set of stream parameters in the Synchronization Parameters text box. These parameters are the same as those in a **stream_parms** parameter, except that a “stream” entry is used to specify the communications type (TCPIP, HTTP, or HTTPS). For example:

```
stream=tcPIP;host=localhost
```

☞ For more information, see “HotSync synchronization stream parameters” [*UltraLite Database User’s Guide*, page 181].

4. If the database is strongly encrypted, you can enter the encryption key in the Encryption Key text box. If no key is entered, you will be prompted for the encryption key on each synchronization.
5. Click OK to complete the entry. The HotSync conduit is now ready to use.

Registry location

Location of
synchronization log files

The stream parameters and encryption key are stored in the registry in *HKEY_CURRENT_USER\Software\Sybase\Adaptive Server Anywhere\9.0\Conduit\Creator ID*, where *Creator ID* is application-dependent.

A secondary location for HotSync synchronization depends on the version of the Palm Computing Platform software you are using. They are made under the *HKEY_CURRENT_USER\Software\U.S. Robotics* or the *HKEY_CURRENT_USER\Software\Palm Computing* folder.

HotSync log files

You can obtain additional debugging information by setting the *UL_DEBUG_CONDUIT_LOG* environment variable. When this environment variable is set to 1, basic information such as synchronization parameters, registry locations, and attempts to load libraries is written to the HotSync log file. When set to 2, more detailed information is written to the HotSync log file.

The HotSync log file is in the subdirectory *User\HotSync.log* of your Pilot or Palm directory. HotSync records when each synchronization takes place and whether each installed conduit worked as expected.

Deploying the MobiLink HotSync conduit

For applications using HotSync synchronization, each end user must have the MobiLink HotSync conduit installed on their desktop. This installation requires the following:

- ◆ **Deploy the conduit files** The files for the conduit must be installed into a location in the end user's system path.
☞ For a list of conduit files, see [“Conduit files” on page 211](#).
- ◆ **Install the conduit** You can deploy the conduit installation utility to your end users and provide instructions for them to run it, or you can use the HotSync Manager to install the conduit.
☞ For instructions, see [“Registering the MobiLink HotSync conduit to HotSync Manager” on page 212](#).
- ◆ **Configure the conduit** If you did not include a **stream_parms** parameter in your UltraLite **ul_synch_info** structure, enter these parameters from the HotSync Manager. Also, if you are using an encrypted database, you may want to enter the encryption key.
☞ For instructions, see [“Configuring the MobiLink HotSync conduit” on page 214](#).

Configuring TCP/IP, HTTP, or HTTPS synchronization

This section describes how to configure the synchronization setup for UltraLite Palm applications using TCP/IP or HTTP synchronization.

☞ For information on synchronization architecture for HTTP or TCP/IP communications, see [“Parts of the synchronization system” on page 8](#).

Configuring TCP/IP synchronization for the Palm Computing Platform

There are two ways of using TCP/IP networking in a Palm device. In either case, you must connect to a Remote Access Service (RAS). The difference lies in how you make the connection to the RAS.

- ◆ **Use a modem to dial into an ISP** The Internet Service Provider (ISP) must provide access to a Remote Access Service (RAS). The components of the connection are as follows:


```
Application
<--> Palm Net Library
<--> Palm modem
<--> NT RAS
<--> TCP/IP network
```

- ◆ **Connect via the serial port to a Windows NT machine** The components of the connection are as follows:

```
Application
<--> Palm Net Library
<--> serial cable
<--> NT RAS
<--> TCP/IP network
```

When using TCP/IP, the MobiLink synchronization server can be any machine on the network that is accessible via TCP/IP.

Before synchronization, the following conditions must be satisfied:

1. The device must be in its cradle.
2. If you are using the serial port to connect to a Windows NT machine running RAS, the HotSync Manager and other applications that use the serial port must be shut down. Windows NT only allows one application to use a serial port at a time.
3. The MobiLink synchronization server must be started. By default, the MobiLink synchronization server listens for TCP/IP communications over port 2439.
4. The Palm device must have Network settings in place so that it can connect to the network. Modem settings are also required if using a modem to dial into an ISP.

Configuring HTTP or HTTPS synchronization for the Palm Computing platform

To use HTTP or HTTPS synchronization, you must first configure RAS TCP/IP synchronization. For information on configuring RAS, see [“Configuring TCP/IP synchronization for the Palm Computing Platform” on page 216](#).

When using HTTP or HTTPS, the MobiLink synchronization server can be any machine on the network that is accessible via the protocol.

❖ To synchronize using HTTP or HTTPS

1. Place the Palm device in its cradle.
2. If you are using the serial port to connect to a desktop machine running RAS, shut down the HotSync Manager and other applications that use the serial port. Windows only allows one application to use a serial port at a time.
3. Start the MobiLink synchronization server.
4. Ensure that the network settings on the Palm device are configured so that it can connect to the network. Modem settings are also required if using a modem to dial into an ISP.

☞ For more information, see [“Configuring TCP/IP synchronization for the Palm Computing Platform” on page 216.](#)

Configuring Remote Access Service

Synchronizing Palm applications over TCP/IP, HTTP, or HTTPS requires a Remote Access Service on your desktop machine. Remote Access Service software is not part of UltraLite or MobiLink. Configuring Remote Access Service software is tricky, however, and so this section provides some instructions to assist with the task.

Configuring RAS for synchronization via modem

To use this method, you must have access to a Remote Access Service when you dial in.

❖ To configure a Palm device for RAS TCP/IP via a modem

1. Install the modem by plugging the Palm device into the modem module.
2. Go to the Preferences (Prefs) panel and choose Network from the dropdown list at the top right of screen.
3. Choose the Windows RAS service.
4. Set the dial-in username and password.
5. Set the phone number to the number at which the Remote Access Service can be reached. Obtain this number from your ISP.
6. Tap Details.
7. Set the connection type (usually PPP).

8. Set the DNS and IP addresses as recommended by your network administrator.
9. Tap Script and enter the script recommended by your ISP. This script will be similar to the following sample.

```
Wait For: Username:
Delay: 1
Send UserID:
Send CR:
Wait For: Password:
Delay: 1
Send Password:
Send CR:
Wait For: >
Delay: 1
Send: ppp
Send CR:
End:
```

Tap OK until you are back to the Network Preferences.

At this point, you are ready to test your TCP/IP connection.

Configuring RAS for serial port connection

This procedure involves actions both on Windows NT and on the Palm Computing device.

❖ To configure Windows NT for RAS TCP/IP via serial port

1. From the Control Panel, open Modems. Make sure that a modem is defined for **Dial-Up Networking Serial Cable between 2 PCs** on the COM port to which the cradle is connected.
2. Set the speed for this modem to the baud rate you are using. The default is 19200.
3. Make sure TCP/IP protocol is installed. Select Start ► Settings ► Control Panel and double-click the Network icon. Click on the Protocols tab. If there is no TCP/IP entry, choose Add to install it.
4. Enable IP Forwarding (in the Routing tab of TCP/IP properties)
5. Under the Services tab, make sure that Remote Access Service is installed. If there is no entry for Remote Access Service, choose Add to install it.

In Remote Access Service Properties, add **Dial Up Network serial cable between 2 pc's** for that COM port if the cradle's COM port is not in the list of ports.

-
6. Configure this entry to receive calls. In the RAS Network properties set encryption settings to **Allow any authentication including clear text**. In the RAS Network properties allow only TCP/IP client.
 7. Configure TCP/IP. Allow clients to access the entire network. Assigning the TCP/IP addresses depends on your network. Contact your network administrator for details.
 8. Add a user for dial-in access. Select Start ► Programs ► Administrative Tools ► User Manager. Uncheck **User Must Change Password at Next Logon**. Choose the Dialin button, and grant dialin permission to user with No Call Back.
 9. If the RAS COM port is the same one that HotSync Manager uses, shut down the HotSync Manager or any other applications that use the COM port.
 10. Start the Remote Access Administrator. Select Start ► Programs ► Administrative Tools ► Remote Access Admin.
 11. Start the RAS service. Select Server ► Start Remote Access Service. Choose to start the service on the local machine.

HotSync Manager or any other applications that use the serial port and the RAS service will not run at the same time. One must be shut down first for the other to run, as Windows NT prevents two different applications from accessing the same serial port. You have to stop the RAS service (Server ► Stop Remote Access Service from the Remote Access Admin) before you can restart the HotSync Manager. Alternatively, you can use separate serial ports.

Once the RAS service is running, it is ready to receive connection requests via the serial port.

❖ To configure a Palm device for RAS TCP/IP via serial port

1. Go to the Preferences (Prefs) panel and choose Network from the dropdown list at the top right of screen.
2. Choose the Windows RAS service.
3. Set the dial-in username and password.
4. Set the Palm to use the serial port.
 - ◆ For Palm OS 3.3 and above, select **Direct serial**.
 - ◆ For earlier versions of the Palm OS, set the phone number to **00** (zero zero). This is a special phone number that tells the Palm to use the serial port directly, instead of a modem.

5. Tap Details.
6. Set the connection type (usually PPP).
7. Set the DNS and IP addresses as recommended by your network administrator.
8. Tap Script and enter the following script:

```
Send: CLIENT
Send CR:
Delay: 1
Send: CLIENT
END
```

Tap OK until you are back to the Network Preferences

At this point, you are ready to test your TCP/IP connection.

Testing and troubleshooting

❖ To test the connection

1. **via modem** Connect the Palm device to the modem and follow the instructions provided by your ISP for connecting to their network. Once connected, tap the Connect button in Prefs ► Network on the Palm device.
2. **via serial port** Ensure RAS is running on the Windows NT machine. Place the Palm device in the cradle and connect the cradle to the correct COM port on the Windows NT machine. Tap the Connect button in Prefs ► Network on the Palm device.

With TCP/IP, there are two levels of service. At the minimum level, you can connect to another TCP/IP host using an IP number of the following form.

```
NNN.NNN.NNN.NNN
```

At the next level, when a DNS server is properly configured, you are able to connect to another host by name.

```
some_host_machine.any_company.com
```

Having a DNS service is more convenient, since most people are better at remembering a name than a number. As long as you have the minimum TCP/IP service, and an IP number, you can synchronize an UltraLite application using TCP/IP.

There are a number of steps you can take to troubleshoot TCP/IP connections on the Palm device.

-
- ◆ Hitting the scroll down button on the Palm device during the connection phase displays the progress of the connection.
 - ◆ The connection log is accessible from the Network Preferences panel. Choose View Log from the Options menu to see information about the network connection. The log is an interactive utility for controlling and viewing your connection information. Enter ? for help.
 - ◆ There are several tools for testing a TCP/IP connection from the Palm. You can find most of them at the following locations:

<http://www.roadcoders.com>
<http://www.palmcentral.com>

There are also steps you can take for troubleshooting on Windows NT:

- ◆ In the Remote Access Admin, double-click on the running server.
- ◆ Select the appropriate port and choose Port Status. The Port Status dialog shows you the Line condition (connected or waiting for a call) and lets you watch the byte counts for both directions.

Synchronizing UltraLite databases on Windows CE

UltraLite applications on Windows CE can synchronize using standard network protocols (TCP/IP, HTTP, or HTTPS). UltraLite applications built using embedded SQL, the static C++ API, or Native UltraLite for Java can also use ActiveSync.

This section describes ActiveSync synchronization.

Installing the MobiLink provider for ActiveSync

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

The MobiLink provider for ActiveSync includes a desktop component and a device component. You must configure the provider for each device that synchronizes through your desktop machine.

When you have installed the MobiLink provider for ActiveSync you must register each application separately. For instructions, see [“Registering applications for use with ActiveSync” on page 224](#).

❖ To install the MobiLink provider for ActiveSync

1. Ensure that you have the ActiveSync software on your machine, and that the Windows CE device is connected.
2. Enter the following command to install the MobiLink provider:

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

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

If you have SQL Anywhere installed on your machine, *dbasdesk.dll* is in the *win32* subdirectory of your SQL Anywhere directory and *dbasdev.dll* is in a platform-specific directory in the *CE* subdirectory. These directories are default search locations, and you can omit both the *-k* and *-v* options.

☞ For more information, see “ActiveSync provider installation utility” [*MobiLink Synchronization Reference*, page 300].

3. Restart your machine.

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

4. Enable the MobiLink provider.
 - ◆ From the ActiveSync window, click Options.
 - ◆ Check the MobiLink item in the list and click OK to activate the provider.
 - ◆ To see a list of registered applications, click Options again, choose the MobiLink provider, and click Settings.
 - ☞ For more information about registering applications, see [“Registering applications for use with ActiveSync” on page 224.](#)

Registering applications for use with ActiveSync

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

☞ For information about the alternative approach, see “ActiveSync provider installation utility” [*MobiLink Synchronization Reference*, page 300].

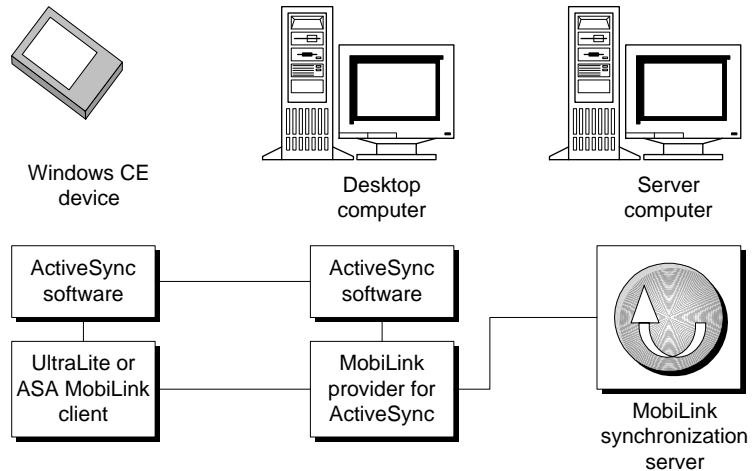
❖ To register an application for use with ActiveSync

1. Ensure that the MobiLink provider for ActiveSync is installed.
 - ☞ For information, see [“Installing the MobiLink provider for ActiveSync” on page 223.](#)
2. Start the ActiveSync software on your desktop machine.
3. From the ActiveSync window, choose Options.
4. From the list of information types, choose MobiLink and click Settings.
5. In the MobiLink Synchronization dialog, click New. The Properties dialog appears.
6. Enter the following information for your application:
 - ◆ **Application name** A name identifying the application to be displayed in the ActiveSync user interface.
 - ◆ **Class name** The registered class name for the application.
 - ☞ See “Assigning class names for applications” [*UltraLite Embedded SQL User’s Guide*, page 93].
 - ◆ **Path** The location of the application on the device.
 - ◆ **Arguments** Any command line arguments to be used when ActiveSync starts the application.

- Click OK to register the application.

Deploying applications that use ActiveSync

Applications that use ActiveSync synchronization must be registered with ActiveSync as well as copied onto the device. Also, each desktop machine must have the MobiLink provider for ActiveSync installed. The architecture for ActiveSync is illustrated in the following diagram.



❖ To deploy ActiveSync applications

- Install the MobiLink provider for ActiveSync on each end user's computer.

An ActiveSync provider install utility is provided with SQL Anywhere. This is the *dbasinst.exe* command line utility.

☞ For information about the *dbasinst.exe* command line utility, see [“Installing the MobiLink provider for ActiveSync” on page 223](#) and [“ActiveSync provider installation utility” \[MobiLink Synchronization Reference, page 300\]](#).

- Register the application for use with ActiveSync.

You can register the application either by using ActiveSync, or by using the ActiveSync provider installation utility *dbasinst.exe*.

☞ For information about registering the application, see [“Registering applications for use with ActiveSync” on page 224](#).

- Copy the application onto the device.

If your application is a single executable, statically linked with the runtime library, you can use the ActiveSync provider installation utility *dbasinst.exe* to copy the application to the device.

If the application includes multiple files (for example, if you use the UltraLite runtime DLL rather than the static runtime library), you must copy the files onto the device in some other way.

CHAPTER 10

Writing Synchronization Scripts in Java

About this chapter

You control the actions of the MobiLink synchronization server by writing synchronization scripts. You can implement these scripts in SQL, .NET or Java. This chapter describes how to implement synchronization scripts in Java.

☞ For a description and comparison of SQL, Java, and .NET, see [“Options for writing synchronization logic” on page 31](#).

☞ For information about writing scripts, see [“Writing Synchronization Scripts” on page 37](#).

☞ For information about writing scripts in .NET, see [“Writing Synchronization Scripts in .NET” on page 251](#).

Contents

Topic:	page
Introduction	228
Setting up Java synchronization logic	229
Running Java synchronization logic	231
Writing Java synchronization logic	232
Java synchronization example	240
MobiLink Java API Reference	246

Introduction

MobiLink synchronization scripts can be written in Java. Java synchronization logic can function just as SQL logic functions: the MobiLink synchronization server can make calls to Java methods on the occurrence of MobiLink events just as it accesses SQL scripts on the occurrence of MobiLink events. A SQL string may be returned as a Java method to MobiLink.

This section tells you how to set up, develop, and run Java synchronization logic. It includes a sample application and the MobiLink Java API.

Setting up Java synchronization logic

When you install SQL Anywhere Studio, the installer automatically sets the location of the MobiLink Java API classes. When you start the MobiLink synchronization server, it automatically includes these classes in your classpath. The MobiLink Java API classes are installed to `Java\mlscript.jar` in your SQL Anywhere Studio installation directory. MobiLink uses the `ASANYSH*` environment variables to determine the shared component path.

☞ For more information, see “`ASANYSH9` environment variable” [ASA Database Administration Guide, page 245].

❖ To implement synchronization scripts in Java

1. Create your own class or classes. Write a method for each required synchronization script. These methods must be public. The class must be public in the package.

☞ For more information about methods, see “[Methods](#)” on page 234.

Each class with non-static methods should have a public constructor. The MobiLink synchronization server automatically instantiates each class the first time a method in that class is called. The class constructor may have one of two signatures, as described below.

☞ For more information about constructors, see “[Constructors](#)” on page 233.

2. When compiling the class, you must include the JAR file `java\mlscript.jar`.

For example,

```
javac MyClass.java -classpath %asany9%\java\mlscript.jar
```

3. In your consolidated database, specify the name of the package, class, and method to call for each script. One class is permitted per script version.

This information is stored in the MobiLink system tables. The `script_language` column of the `ml_script` table must contain the word **java**. The string in the `script` column, which contains a statement for scripts implemented in SQL, must instead contain the qualified name of a public Java method.

The easiest way to add this information to the MobiLink system tables is to use the `ml_add_java_connection_script` stored procedure or the `ml_add_java_table_script` stored procedure. You can also add this information using Sybase Central.

☞ For more information, see “ml_add_java_connection_script” [MobiLink Synchronization Reference, page 266] and “ml_add_java_table_script” [MobiLink Synchronization Reference, page 267].

For example, the following statement, when run in an Adaptive Server Anywhere database, specifies that myPackage.myClass.myMethod should be run whenever the authenticate_user connection-level event occurs.

```
call ml_add_java_connection_script( 'version1',  
    'authenticate_user', 'myPackage.myClass.myMethod' )
```

The example code below, for use with the Sample application, calls Java procedures to add connection and table script data to the MobiLink system tables.

```
call ml_add_java_connection_script('ver1', 'authenticate_  
    user',  
    'CustEmpScripts.authenticateUser')  
call ml_add_java_connection_script('ver1', 'end_connection',  
    'CustEmpScripts.endConnection')  
call ml_add_java_table_script('ver1', 'emp', 'upload_  
    cursor',  
    'CustEmpScripts.empUploadCursor')  
call ml_add_java_table_script('ver1', 'emp',  
    'download_cursor', 'CustEmpScripts.empDownloadCursor')  
call ml_add_java_table_script('ver1', 'cust', 'upload_  
    cursor',  
    'CustEmpScripts.custUploadCursor')  
call ml_add_java_table_script('ver1', 'cust',  
    'download_cursor',  
    'CustEmpScripts.custDownloadCursor')
```

4. A vital part of setting up for using Java synchronization logic is to establish a **classpath** to tell the virtual machine where to look for Java classes. There are two ways to do this:

- ◆ To set the classpath for user-defined classes, use a statement such as the following:

```
SET classpath=%classpath%;c:\local\Java\myclasses.jar
```

- ◆ Use the dbmlsrv9 -sl java -cp option to specify a set of directories or jar files in which to search for classes.

☞ For more information about -cp, see “-sl java option” [MobiLink Synchronization Reference, page 19].

Running Java synchronization logic

If your system classpath includes your Java synchronization logic classes, you do not need to make changes to your MobiLink synchronization server command line.

You can use the `-sl java` option to force the Java virtual machine to load at server startup. Otherwise, the Java virtual machine is started when the first Java method is executed.

```
dbmlsrv9 -c "dsn=MyDataSource" -sl java ...
```

You also can set the classpath and pass flags to the Java virtual machine on the MobiLink synchronization server command line. The MobiLink synchronization server automatically appends the location of the MobiLink Java API classes (`java\mlscript.jar`) to your classpath.

```
dbmlsrv9 -c "dsn=MyDataSource" -sl java ( -cp c:\local\Java\  
myclass.jar )
```

☞ For more information about the available Java options, see “`-sl java` option” [*MobiLink Synchronization Reference*, page 19].

Writing Java synchronization logic

Writing Java synchronization logic is no different in complexity from writing any other Java code. What is required from you is knowledge of MobiLink events, some knowledge of Java, and knowledge of the MobiLink Java API. The following sections help you write useful synchronization logic.

In this release, the row data for the upload and download streams are not passed to the Java synchronization logic. Java synchronization logic can be used to maintain state information, and implement logic around the upload and download events. For example, a `begin_synchronization` script written in Java could store the MobiLink user ID in a variable. Scripts called later in the synchronization process can access this variable.

Using Java reduces dependence on the consolidated database. Behavior is affected less by upgrading the consolidated database to a new version or switching to a different database-management system.

☞ For a complete description of the API, see [“MobiLink Java API Reference” on page 246](#).

Class instances

The MobiLink synchronization server instantiates your classes at the connection level. When an event is reached for which you have written a non-static Java method, the MobiLink synchronization server automatically constructs the class, if it has not already done so on the present connection. To do so, it uses the class constructor.

For each database connection, once a class has been instantiated, the class persists until that connection is closed. Thus, the same instance may well be used for multiple consecutive synchronization sessions. Information present in public or private variables will thus persist across synchronizations that occur on the same connection unless explicitly cleared.

You can also use static classes or variables. In this case, the values are available across all connections.

The MobiLink synchronization server automatically deletes your class instances only when the connection to the consolidated database is closed.

All methods in one script version called on the same connection *must belong to the same class*.

Transactions

The normal rules regarding transactions apply to Java methods. The start and duration of database transactions is critical to the synchronization

process. Transactions must be started and ended only by the MobiLink synchronization server. Explicitly committing or rolling back transactions on the synchronization connection within a Java method violates the integrity of the synchronization process and can cause errors.

These rules apply only to the database connections created by the MobiLink synchronization server and, in particular, to SQL statements returned by methods. If your classes create other database connections, you are free to manage them as you please.

SQL-Java data types

The following table shows SQL data types and the corresponding Java data types.

SQL data type	Corresponding Java data type
VARCHAR	java.lang.String
CHAR	java.lang.String
INTEGER	Int or Integer
BINARY	byte[]
TIMESTAMP	java.sql.Timestamp
INOUT INTEGER	ianywhere.ml.script.InOutInteger
INOUT VARCHAR	ianywhere.ml.script.InOutString
INOUT CHAR	ianywhere.ml.script.InOutString
INOUT BYTEARRAY	ianywhere.ml.script.InOutByteArray

The MobiLink synchronization server automatically adds this package to your classpath if it is not already present.

However, when you compile your class you need to add the path of `java\mlscript.jar`, from your SQL Anywhere Studio installation directory.

Constructors

The constructor of your class may have one of two possible signatures.

```
public MyScriptClass (
    ianywhere.ml.script.DBConnectionContext sc )
```

or

```
public MyScriptClass ( )
```

The synchronization context passed to you is for the connection through which the MobiLink synchronization server is synchronizing the current user.

The `getConnection` method of the `DBConnectionContext` returns the same database connection that MobiLink is using to synchronize the present user. You can execute statements on this connection, but you must not commit or roll back the transaction. The MobiLink synchronization server manages the transactions.

The MobiLink synchronization server prefers to use constructors with the first signature. It only uses the non-argument constructor if a constructor with the first signature is not present.

Methods

In general, you implement one method for each synchronization event. These methods must be public. If they are private, the MobiLink synchronization server cannot use them and will fail to recognize that they exist.

The names of the methods are not important, as long as the names match the names specified in the `ml_script` table in the consolidated database. In the examples included in the documentation, however, the method names are the same as those of the MobiLink events as this naming convention makes the Java code easier to read.

The signature of your method should match the signature of the script for that event, except that you can truncate the parameter list if you do not need the values of parameters at the end of the list. Indeed, you should accept only the parameters you need, because overhead is associated with passing the parameters.

You cannot, however, overload the methods. In other words, you must provide *only one method* per class with the name specified in the `ml_script` table.

Return values

Methods called for a MobiLink upload or download must return a valid SQL-language statement. The return type of these methods must be `java.lang.String`. No other return types are allowed.

The return type of all other scripts must either be `java.lang.String` or `void`. No other types are allowed. If the return type is a string and not null, the MobiLink synchronization server assumes that the string contains a valid SQL statement and executes this statement in the consolidated database as it would an ordinary SQL-language synchronization script. If a method ordinarily returns a string but does not wish to execute a SQL statement

against the database upon its return, it can return null.

Debugging Java classes

MobiLink provides various information and facilities that you may find helpful when debugging your Java code. This section describes where you can find this information and exploit these capabilities.

Information in the
MobiLink synchronization
server's log file

The MobiLink synchronization server writes various related information to its output log file. The synchronization server log file contains the following information:

- ◆ The Java Runtime Environment. You can use the `-jrepath` option to request a particular JRE when you start the MobiLink synchronization server. The default is the path installed with Adaptive Server Anywhere 9.
 - ◆ The path of the standard Java classes loaded. If you did not specify these explicitly, the MobiLink synchronization server automatically adds them to your classpath before invoking the Java virtual machine.
 - ◆ The fully specified names of the specific methods invoked. You can use this information to verify that the MobiLink synchronization server is invoking the correct methods.
 - ◆ Any output written in a Java method to `java.lang.System.out` or `java.lang.System.err` is redirected to the MobiLink synchronization server log file.
 - ◆ The `dbmlsrv9` command line option `-verbose` can be used.
- ☞ For more information, see “`-sl java option`” [*MobiLink Synchronization Reference*, page 19].

Using a Java debugger

You can debug your Java classes using a standard Java debugger. Specify the necessary parameters using the `-sl java` option on the `dbmlsrv9` command line.

☞ For more information, see “`-sl java option`” [*MobiLink Synchronization Reference*, page 19].

Specifying a debugger causes the Java virtual machine to pause and wait for a connection from a Java debugger.

Printing information from
Java

Alternatively, you may choose to add statements to your Java methods that print information to the MobiLink output log, using `java.lang.System.err` or `java.lang.System.out`. Doing so can help you track the progress and behavior of your classes.

Performance tip

Printing information in this manner is a useful monitoring tool, but is not recommended in a production scenario.

The same technique can be exploited to log arbitrary synchronization information or collect statistical information on how your scripts are used.

Writing your own test driver

You may wish to write your own driver to exercise your Java classes. This approach can be helpful because it isolates the actions of your Java methods from the rest of the MobiLink system.

Handling MobiLink server errors in Java

When scanning the log is not sufficient, you can monitor your applications programmatically. For example, you can send messages of a certain type in an email.

You can write methods that are passed a class representing every error or warning message that is printed to the log. This may help you monitor and audit a MobiLink synchronization server.

The following code installs a LogListener for all warning messages, and writes the information to a file.

```
class TestLogListener implements LogListener {
    FileOutputStream    _out_file;

    public TestLogListener(    FileOutputStream    out_file )
    {
        _out_file        = out_file;
    }

    public void messageLogged(    ServerContext    sc,
                                LogMessage        msg )
    {
        String    type;
        String    user;
        try {
            if(msg.getType() == LogMessage.ERROR) {
                type = "ERROR";
            } else if(msg.getType() == LogMessage.WARNING)
            {
                type = "WARNING";
            } else {
                type = "UNKNOWN!!!";
            }
        }
    }
}
```

```

        user = msg.getUser();
        if( user == null ) {
            user = "NULL";
        }
        _out_file.write(
            ("Caught msg type=" + type +
             " user=" + user +
             " text=" +msg.getText() +
             "\n").getBytes() );
        _out_file.flush();
    } catch( Exception e ) {
        // print some error output to the MobiLink log
        e.printStackTrace();
    }
}

// This line of code will register TestLogListener to receive
// warning messages. Call this code from anywhere that has
// access to the ServerContext such as a class constructor or
// synchronization script. ServerContext serv_context;
serv_context.addWarningListener(
    new MyLogListener( ll_out_file ));

```

See also

addErrorListener, removeErrorListener, addWarningListener,
removeWarningListener in [“ServerContext interface” on page 248](#)
[“LogListener interface” on page 247](#)
[“LogMessage class” on page 247](#)

User-defined start classes

You can define start classes that are loaded automatically when the server is started. The purpose of this feature is to allow you to write Java code that executes at the time the MobiLink server starts the JVM—before the first synchronization. This means you can create connections or cache data before a user synchronization request.

You do this with the DMLStartClasses option of the dbmlsrv9 -sl java option. For example, the following is part of a dbmlsrv9 command line. It causes mycl1 and mycl2 to be loaded as start classes.

```
-sl java(-DMLStartClasses=com.test.mycl1,com.test.mycl2)
```

Classes are loaded in the order in which they are listed. If the same class is listed more than once, more than one instance is created.

All start classes must be public and must have a public constructor that either accepts no arguments or accepts one argument of type `ianywhere.ml.script.ServerContext`.

The names of loaded start classes are output to the MobiLink log with the message “Loaded JAVA start class: *classname*”.

☞ For more information about Java virtual machine options, see “-sl java option” [*MobiLink Synchronization Reference*, page 19].

☞ To see the start classes that are constructed at server start time, see “[getStartClassInstances](#)” on page 248.

Example

Following is a template start class. It starts a daemon thread that processes events and creates a database connection. (Not all start classes will need to create a thread but if a thread is spawned it should be a daemon thread.)

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

public class StartTemplate extends
    Thread implements ShutdownListener {
//=====
    ServerContext    _sc;
    Connection       _conn;
    boolean          _exit_loop;

    public StartTemplate( ServerContext sc )
//=====
        throws SQLException
    {
        // perform setup first so that an exception will
        // cause MobiLink startup to fail
        _sc      = sc;
        // create a connection for use later
        _conn    = _sc.makeConnection();
        _exit_loop = false;
        setDaemon( true );
        start();
    }
}
```

```
public void run()
//=====
{
    _sc.addShutdownListener( this );
    // we can't throw any exceptions through run()
    try {
        handlerLoop();
        _conn.close();
        _conn = null;
    } catch( Exception e ) {
        // print some error output to the MobiLink log
        e.printStackTrace();
        // we will die so we don't need to be notified
        // of shutdown
        _sc.removeShutdownListener( this );
        // ask server to shutdown so that this fatal
        // error will be fixed
        _sc.shutdown();
    }
    // shortly after return this thread will no longer
    // exist
    return;
}

public void shutdownPerformed( ServerContext sc )
//=====
// stop our event handler loop
{
    try {
        // wait max 10 seconds for thread to die
        join( 10*1000);
    } catch( Exception e ) {
        // print some error output to the MobiLink log
        e.printStackTrace();
    }
}


private void handlerLoop()
//=====throws InterruptedException
{
    while( !_exit_loop ) {
        // Handle events in this loop. Sleep not
        // needed, we should block on event queue.
        sleep( 1*1000 );
    }
}
}
```

Java synchronization example

Java synchronization logic works with MobiLink and common Java classes to provide you with flexibility in deploying applications using MobiLink synchronization server. The following section introduces you to this extended range of functionality using a simple example.

Introduction

This section describes a working example of Java synchronization logic. Before you try to use this class or write your own class, use the following checklist to ensure you have all the pieces in place before compiling the class.

- ◆ Plan your desired functionality using, for example, pseudocode.
- ◆ Create a map of database tables and columns.
- ◆ Set up the consolidated database by ensuring you have specified in the MobiLink system tables the language type and location of the Java synchronization methods.
 For more information see [“Setting up Java synchronization logic” on page 229](#).
- ◆ Create a list of associated Java classes that are called during the running of your Java class.
- ◆ Have a location for your Java classes that is in the classpath for MobiLink synchronization server.

Plan

The Java synchronization logic for this example points to the associated Java files and classes that contain functionality needed for the example to work. It will show you how to create a class `CustEmpScripts`. It shows you how to set up a synchronization context for the connection. Finally, the example provides Java methods to

- ◆ Authenticate a MobiLink user
- ◆ Perform download and upload operations using cursors for each database table.

Schema

The tables to be synchronized are `emp` and `cust`. The `emp` table has three columns called `emp_id`, `emp_name` and `manager`. The `cust` table has three columns called `cust_id`, `cust_name` and `emp_id`. All columns in each table are synchronized. The mapping from consolidated to remote database is such that the table names and column names are identical in both databases. One additional table, an audit table, is added to the consolidated database.

Java class files

The files used in the example are included in the *Samples\MobiLink\JavaAuthentication* directory.

Create your Java synchronization script

Setup

The following sets up the Java synchronization logic. The import statements tell the Java virtual machine the location of needed files. The public class statement declares the class.

```
// use a package when you create your own script
import   ianywhere.ml.script.InOutInteger;
import   ianywhere.ml.script.DBConnectionContext;
import   ianywhere.ml.script.ServerContext;
import   java.sql.*;
public class CustEmpScripts {
/* Context for this synchronization connection.
*/
    DBConnectionContext _conn_context;
/* Same connection MobiLink uses for sync we can't commit or
 * close this.
*/
    Connection          _sync_connection;
    Connection          _audit_connection;
/* Get a user id given the user name. On audit connection.
*/
    PreparedStatement    _get_user_id_pstmt;
/* Add record of user logins added. On audit connection.
*/
    PreparedStatement    _insert_login_pstmt;
/* Prepared statement to add a record to the audit table.
 * On audit connection.
*/
    PreparedStatement    _insert_audit_pstmt;
```

The *CustEmpScripts* constructor sets up all the prepared statements for the *authenticateUser* method. It sets up member data.

```

public CustEmpScripts( DBConnectionContext cc )
    throws SQLException
{
    try
    {
        _conn_context = cc;
        _sync_connection = _conn_context.getConnection();

        ServerContext serv_context =
            _conn_context.getServerContext();
        _audit_connection = serv_context.makeConnection();

        // get the prep statements ready
        _get_user_id_pstmt =
            _audit_connection.prepareStatement(
                "select user_id from ml_user where name = ?"
            );
        _insert_login_pstmt =

        _audit_connection.prepareStatement(
            "insert into login_added( ml_user, add_time )
            " + " values( ?, { fn CONVERT( { fn NOW() },
            SQL_VARCHAR ) } )"
        );
        _insert_audit_pstmt =
            _audit_connection.prepareStatement(
                "insert into login_audit( ml_user_id,
                audit_time, audit_action ) " +
                " values( ?, { fn CONVERT( { fn NOW() },
                SQL_VARCHAR ) }, ? ) "
            );
    } catch ( SQLException e ) {
        freeJDBCResources();
        throw e;
    } catch ( Error e ) {
        freeJDBCResources();
        throw e;
    }
}

```

The finalize method cleans up JDBC resources if end_connection is not called.

```

protected void finalize()
    throws SQLException, Throwable
{
    super.finalize();
    freeJDBCResources();
}

```

The freeJDBCResources method frees allocated memory and we close the audit connection. It is a housekeeping procedure.

```
private void freeJDBCResources()
    throws SQLException
{
    if( _get_user_id_pstmt != null ) {
        _get_user_id_pstmt.close();
    }
    if( _insert_login_pstmt != null ) {
        _insert_login_pstmt.close();
    }
    if( _insert_audit_pstmt != null ) {
        _insert_audit_pstmt.close();
    }
    if( _audit_connection != null ) {
        _audit_connection.close();
    }
    _conn_context      = null;
    _sync_connection   = null;
    _audit_connection  = null;
    _get_user_id_pstmt = null;
    _insert_login_pstmt = null;
    _insert_audit_pstmt = null;
}
```

The `endConnection` method cleans up resources once the resources are not needed. This is also a housekeeping procedure.

```
public void endConnection()
    throws SQLException
{
    freeJDBCResources();
}
```

The `authenticateUser` method below approves all user logins and logs user information to database tables. If the user is not in the `ml_user` table they are logged to `login_added`. If the user id is found in `ml_user` then they are logged to `login_audit`. In a real system we would not ignore the `user_password` but in order to keep this sample simple we approve all users. The procedure throws `SQLException` if any of the database operations we perform fail with an exception

```

public void authenticateUser( InOutInteger auth_status,
    String user_name )
    throws SQLException
{
    boolean new_user;
    int user_id;

    // get ml_user id
    _get_user_id_pstmt.setString( 1, user_name );
    ResultSet user_id_rs =
        _get_user_id_pstmt.executeQuery();
    new_user = !user_id_rs.next();
    if( !new_user ) {
        user_id = user_id_rs.getInt(1);
    } else {
        user_id = 0;
    }

    user_id_rs.close();
    user_id_rs = null;
    // in this tutorial always allow the login
    auth_status.setValue( 1000 );
    if( new_user ) {
        _insert_login_pstmt.setString( 1, user_name );
        _insert_login_pstmt.executeUpdate();
        java.lang.System.out.println( "user: " +
            user_name + " added. " );
    } else {
        _insert_audit_pstmt.setInt( 1, user_id );
        _insert_audit_pstmt.setString( 2, "LOGIN ALLOWED" );
        _insert_audit_pstmt.executeUpdate();
    }
    _audit_connection.commit();
    return;
}

```

The following methods use SQL code to act as cursors on the database tables. Since these are cursor scripts, they must return a SQL string.

```

public static String empUploadInsertStmt()
{
    return( "INSERT INTO emp(
        emp_id, emp_name) VALUES( ?, ?) " );
}

public static String empUploadDeleteStmt()
{
    return( "DELETE FROM emp WHERE emp_id = ?" );
}

public static String empUploadUpdateStmt()
{
    return( "UPDATE emp SET emp_name = ?
        WHERE emp_id = ? " );
}

```

```
public static String empDownloadCursor()
{
    return( "SELECT emp_id, emp_name FROM emp" );
}

public static String custUploadInsertStmt()
{
    return( "INSERT INTO cust(
        cust_id, emp_id, cust_name)
        VALUES ( ?, ?, ? ) " );
}

public static String custUploadDeleteStmt()
{
    return( "DELETE FROM cust WHERE cust_id = ? " );
}

public static String custUploadUpdateStmt()
{
    return( "UPDATE cust
        SET emp_id = ?, cust_name = ?
        WHERE cust_id = ? " );
}

public static String custDownloadCursor()
{
    return( "SELECT cust_id, emp_id, cust_name
        FROM cust" );
}
}
```

This code would be compiled using the command

```
javac -cp %asany9%\java\mlscript.jar CustEmpScripts.jar
```

and we could run the MobiLink synchronization server with the location of CustEmpScripts.class in the classpath. Following is a partial command line:

```
dbmlsrv9 ... -sl java (-cp <class_location>)
```

MobiLink Java API Reference

This section explains the MobiLink Java interfaces and classes, and their associated methods and constructors.

DBConnectionContext interface

An encapsulation of context that lives for the duration of one database connection. A DBConnectionContext is not valid outside of the thread that calls into the user written Java code. If context is required for a background thread or beyond the lifetime of a connection use a ServerContext.

getConnection method `public java.sql.Connection getConnection()`
throws `java.sql.SQLException`

Returns the existing connection `java.sql.Connection` as a JDBC connection. The connection is the same connection that MobiLink uses to execute SQL scripts.

This connection must not be committed, closed or altered in any way that would affect the MobiLink server use of this connection. The connection returned is only valid for the lifetime of the underlying MobiLink connection. Do not use the connection after the `end_connection` event has been called for that connection.

If an error occurs binding the existing connection as a JDBC connection then it throws `java.sql.SQLException`

If a server connection with full access is required, use `ServerContext.makeConnection()`.

getServerContext method `public ServerContext getServerContext()`

Returns the ServerContext for this MobiLink server.

InOutByteArray interface

Passed into methods to enable the functionality of an in/out parameter passed to a SQL script.

getValue method `public byte[] getValue()`

Returns the value of this byte array parameter.

setValue method `public void setValue() byte[] new_value`

Sets the value of this byte array parameter. There is one parameter, *new_value*, which is the value this byte array should take.

InOutInteger interface

Passed into methods to enable the functionality of an in/out parameter passed to a SQL script.

getValue method `public int getValue()`

Returns the value of this integer parameter.

setValue method `public void setValue(int new_value)`

Sets the value of this integer parameter. There is one parameter, *new_value*, which is the value this integer should take.

InOutString interface

Passed into methods to enable the functionality of an in/out parameter passed to a SQL script.

getValue method `public java.lang.String getValue()`

Returns the value of this string parameter.

setValue method `public void setValue(int new_value)`

Sets the value of this integer parameter. There is one parameter, *new_value*, which is the value this string should take.

LogListener interface

The listener interface for catching messages that are printed to the log.

messageLogged method `public void messageLogged(
 ServerContext sc
 LogMessage message)`

Invoked when a message is printed to the log. There are two parameters: *sc*, which is the context for the server that is printing the message; and *message*, which is the LogMessage that has been sent to the MobiLink log.

LogMessage class

Holds the data associated with a log message.

Extends java.lang.Object

Constants `int ERROR`

	int WARNING
getType method	<pre>public int getType()</pre> <p>Accessor for this message type.</p> <p>Returns the type of this message, which can be either ERROR or WARNING.</p>
getUser method	<pre>public java.lang.String getUser()</pre> <p>Accessor for this message user. If the message has no user, then the user is NULL.</p> <p>Returns the user associated with this message.</p>
getText method	<pre>public java.lang.String getText()</pre> <p>Accessor for the message text.</p> <p>Returns the main text of this message.</p>

ServerContext interface

An instantiation of all the context that is present for the duration of the MobiLink server. This context can be held as static data and used in a background thread. It is valid for the duration of the Java virtual machine invoked by MobiLink.

addShutdownListener	<pre>public void addShutdownListener (ShutdownListener s/)</pre> <p>Adds the specified ShutdownListener that is to receive notification before the server context is destroyed. On shutdown, the method ShutdownListener.shutdownPerformed (ianywhere.ml.script.ServerContext) is called. There is one parameter, <i>s/</i>, which specifies that the ShutdownListener is to be notified on shutdown.</p>
removeShutdownListener	<pre>public void removeShutdownListener (ShutdownListener s/)</pre> <p>Removes the specified ShutdownListener from the list of listeners that are to receive notification before the server context is destroyed. There is one parameter, <i>s/</i>, which specifies the listener that will no longer be notified on shutdown.</p>
shutdown	<pre>public void shutdown()</pre> <p>Forces the server to shut down.</p>
getStartClassInstances	<pre>public java.lang.Object[] getStartClassInstances()</pre>

Gets an array of the start classes that were constructed at server start time. The array length is zero if there are no start classes.

☞ For more information about user-defined start classes, see [“User-defined start classes” on page 237](#).

Following is an example of `getStartClassInstances()`:

```
Object objs[] = sc.getStartClassInstances();
int i;
for( i=0; i<objs.length; i+=1 ) {
    if( objs[i] instanceof MyClass ) {
        //use class
    }
}
```

makeConnection method `public java.sql.Connection makeConnection()`
throws `java.sql.SQLException`

Opens and returns a new server connection. To access the server context, use `DBConnectionContext.getServerContext` on the synchronization context for the current connection. If an error occurs opening a new connection, the method throws `java.sql.SQLException`.

addErrorListener method `public void addErrorListener(LogListener //)`

Adds the specified `LogListener` to receive a notification when an error is printed.

// is the `LogListener` that is to be notified.

The following method will be called:

`LogListener.messageLogged(ianywhere.ml.script.ServerContext, ianywhere.ml.script.LogMessage).`

removeError Listener method `public void removeErrorListener(LogListener //)`

Removes the specified `LogListener` from the list of listeners that are to receive a notification when an error is printed.

// is the `LogListener` that is no longer to be notified.

addWarningListener method `public void addWarningListener(LogListener //)`

Adds the specified `LogListener` to receive a notification when a warning is printed.

// is the `LogListener` that is to be notified.

The following method will be called:

`LogListener.messageLogged(ianywhere.ml.script.ServerContext, ianywhere.ml.script.LogMessage).`

removeWarningListener method	<pre>public void removeWarningListener(LogListener //)</pre> <p>Removes the specified LogListener from the list of listeners that are to receive a notification when a warning is printed.</p> <p>// is the LogListener that is no longer to be notified.</p>
---------------------------------	--

ServerException class

Thrown to indicate that there is an error condition that makes any further synchronization on the server impossible. Throwing this exception causes the MobiLink server to shut down.

ServerException constructors	<pre>public ServerException()</pre> <p>Constructs a ServerException with no detail message.</p> <pre>public ServerException(java.lang.String s)</pre> <p>Constructs a ServerException with a specified detail message. There is one parameter, s, which specifies the detailed message.</p>
---------------------------------	--

ShutdownListener interface

The listener interface for catching server shutdowns. Use this interface to ensure that all resources, threads, connections, and so on are cleaned up before the server exits.

shutdownPerformed method	<pre>public void shutdownPerformed(ServerContext sc)</pre> <p>Invoked before the ServerContext is destroyed due to server shutdown. There is one parameter, sc, which is the context for the server that is being shut down.</p>
-----------------------------	--

SynchronizationException class

Thrown to indicate that there is an error condition that makes the completion of the current synchronization impossible. Throwing this exception will force the MobiLink server to rollback.

SynchronizationException constructors	<pre>public SynchronizationException()</pre> <p>Constructs a SynchronizationException with no detail message.</p> <pre>public SynchronizationException(java.lang.String s)</pre> <p>Constructs a SynchronizationException with the specified detail message. There is one parameter, s, which specifies a detail message.</p>
--	--

CHAPTER 11

Writing Synchronization Scripts in .NET

About this chapter

You control the actions of the MobiLink synchronization server by writing synchronization scripts. You can implement these scripts in SQL, Java, or .NET. This chapter describes how to implement synchronization scripts in .NET.

☞ For information about writing scripts, see [“Writing Synchronization Scripts” on page 37](#).

☞ For a description and comparison of SQL, Java, and .NET, see [“Options for writing synchronization logic” on page 31](#).

☞ For information about writing scripts in Java, see [“Writing Synchronization Scripts in Java” on page 227](#).

Contents

Topic:	page
Introduction	252
Setting up .NET synchronization logic	253
Running .NET synchronization logic	255
Writing .NET synchronization logic	260
.NET synchronization example	266
MobiLink .NET API Reference	269

Introduction

☞ Microsoft .NET is a platform for building, deploying, and running Web services and applications.

☞ MobiLink supports Visual Studio .NET programming languages for writing synchronization scripts. To write MobiLink scripts in .NET, you can use any language that lets you create valid .NET assemblies. In particular, the following languages are tested and documented:

- ◆ C#
- ◆ Visual Basic .NET
- ◆ C++

.NET synchronization logic can function just as Java logic functions: the MobiLink synchronization server can make calls to .NET methods on the occurrence of MobiLink events. A SQL string may be returned to MobiLink.

This section tells you how to set up, develop, and run .NET synchronization logic for C#, Visual Basic .NET, and C++. It includes a sample application and the MobiLink .NET API Reference.

Setting up .NET synchronization logic

☞ The most important part of implementing synchronization scripts in .NET is telling MobiLink where to find the packages, classes, and methods that are contained in your assemblies. This is described, below.

❖ To implement synchronization scripts in .NET

1. Create your own class or classes. Write a method for each required synchronization event. These methods must be public.

☞ For more information about methods, see [“Methods” on page 262](#).

Each class with non-static methods should have a public constructor. The MobiLink synchronization server automatically instantiates each class the first time a method in that class is called for a connection. The class constructor may have one of two signatures, as described below.

☞ For more information about constructors, see [“Constructors” on page 261](#).

2. In the MobiLink system tables in your consolidated database, specify the name of the package, class, and method to call for each script. Optionally, specify the domain. One class is permitted per script version.

The script_language column of the ml_script system table must contain the word **dnet**. The string in the script column, which contains a statement for scripts implemented in SQL, must instead contain the qualified name of a public .NET method.

The easiest way to add this information to the MobiLink system tables is to use the ml_add_dnet_connection_script stored procedure or the ml_add_dnet_table_script stored procedure. You can also add this information using Sybase Central.

☞ For more information, see [“ml_add_dnet_connection_script”](#) [*MobiLink Synchronization Reference*, page 264] and [“ml_add_dnet_table_script”](#) [*MobiLink Synchronization Reference*, page 265].

For example, the following statement, when run in an Adaptive Server Anywhere database, specifies that myNamespace.myClass.myMethod should be run whenever the authenticate_user connection-level event occurs.

```
call ml_add_dnet_connection_script( 'version1',
  'authenticate_user', 'myNamespace.myClass.myMethod' )
```

3. Create one or more assemblies. You tell MobiLink where to locate these assemblies using options on the dbmlsrv9 command line. There are two options to choose from:

-
- ◆ **Use -sl dnet (-MLAutoLoadPath)** This sets the given path to the application base directory and loads all the assemblies within it. You should use this option in most cases.
 - ◆ **Use -sl dnet (-MLDomConfigFile)** This option requires a configuration file that contains domain and assembly settings. You should use this option when you have shared assemblies, when you don't want to load all the assemblies in a directory, or when for some other reason you need to use a configuration file.
 - ☞ For more information about loading assemblies, see [“Loading assemblies” on page 255](#). For more information about the dbmlsrv9 option -sl dnet, see “-sl dnet option” [*MobiLink Synchronization Reference*, page 17].

Running .NET synchronization logic

This section describes how to run .NET synchronization logic.

Loading assemblies

A .NET assembly is a package of types, metadata, and executable code. In .NET applications, all code must be in an assembly. Assembly files have the extension *.dll* or *.exe*.

There are two types of assembly:

- ◆ **Private assemblies** A private assembly is a file in the file system.
- ◆ **Shared assemblies** A shared assembly is an assembly that is installed in the global assembly cache.

Before MobiLink can load a class and call a method of that class, it must locate the assembly that contains the class. MobiLink only needs to locate the assembly that it calls directly. The assembly can then call any other assemblies it needs.

For example, MobiLink calls *MyAssembly*, and *MyAssembly* calls *UtilityAssembly* and *NetworkingUtilsAssembly*. In this situation, MobiLink only needs to be configured to find *MyAssembly*.

MobiLink provides two ways to load assemblies:

- ◆ **Use `-sl dnet (-MLAutoLoadPath)`** This option only works with private assemblies. It sets the path to the application base directory and loads all the assemblies within it. This option is simpler to use and it is expected that it will be sufficient in most cases.

When you use this option, you cannot specify a domain in the event script.

When you specify a path and directory with `-MLAutoLoadPath`, MobiLink does the following:

- sets this path as the application base path
- loads all classes in all files ending with *.dll* or *.exe* in the directory that you specified
- creates one application domain and loads into that domain all user classes that do not have a domain specified

Assemblies in the global assembly cache cannot be called directly with this option. To call these shared assemblies, use `-MLDomConfigFile`.

-
- ◆ **Use -sl dnet (-MLDomConfigFile)** This option works with both private and shared assemblies. It requires a configuration file that contains domain and assembly settings. You should use this option when you have shared assemblies, when you don't want to load all the assemblies in the application base path, or when for some other reason you need to use a configuration file.

With this option, MobiLink reads the settings in the specified domain configuration file. A domain configuration file contains configuration settings for one or more .NET domains. If there is more than one domain represented in the file, the first one that is specified is used as the default domain. (The default domain is used when scripts do not have a domain specified.)

Only assemblies that are specified in the domain configuration file can be called directly from event scripts.

When loading assemblies, MobiLink tries to load the assembly first as private, and then attempts to load the assembly from the global assembly cache. Private assemblies must be located in the application base directory. Shared assemblies are loaded from the global assembly cache.

Sample domain
configuration file

A sample domain configuration file is installed with MobiLink. You can write your own file from scratch, or edit the sample to suit your needs. The sample file is located in the SQL Anywhere Studio path, in

MobiLink\setup\dnet\mlDomConfig.xml

Following is the content of the sample domain configuration file *mlDomConfig.xml*:

```
<?xml version="1.0" encoding="utf-8"?>
<config xmlns="iAnywhere.MobiLink.mlDomConfig"
        xsi:schemaLocation='iAnywhere.MobiLink.mlDomConfig
        mlDomConfig.xsd'
        xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' >
  <domain>
    <name>SampleDomain1</name>
    <appBase>C:\scriptsDir</appBase>
    <configFile></configFile>
    <assembly name="Assembly1" />
    <assembly name="Assembly2" />
  </domain>
  <domain>
    <name>SampleDomain2</name>
    <appBase>\Dom2assembly</appBase>
    <configFile>\Dom2assembly\
      AssemblyRedirects.config</configFile>
    <assembly name="Assembly3" />
    <assembly name="Assembly4" />
  </domain>
</config>
```


Following is an explanation of the contents of *mlDomConfig.xml*:

- ◆ **name** is the domain name, used when specifying the domain in an event script. An event script with the format `"DomainName:Namespace.Class.Method"` would require a domain called `DomainName` be in the domain configuration file.
You must specify at least one domain name.
- ◆ **appBase** is the directory that the domain should use as its application base directory. All private assemblies are loaded by the .NET CLR based on this directory. You must specify `appBase`.
- ◆ **configFile** is the .NET application configuration file that should be used for the domain. This can be left blank. It is usually used to modify the default assembly binding and loading behavior. Refer to your .NET documentation for more information about application configuration files.
- ◆ **assembly** is the name of an assembly that MobiLink should load and search when resolving type references in event scripts. You must specify at least one assembly. If an assembly is used in more than one domain, it must be specified as an assembly in each domain. If the assembly is private, it must be in the application base directory for the domain.

☞ For more information about the `dbmlsrv9` option `-sl dnet`, see “`-sl dnet` option” [*MobiLink Synchronization Reference*, page 17].

Printing information from .NET

You may choose to add statements to your .NET methods that print information to the MobiLink log using `System.Console`. Doing so can help you track the progress and behavior of your classes.

Performance tip

Printing information in this manner is a useful monitoring tool, but is not recommended in a production scenario.

The same technique can be exploited to log arbitrary synchronization information or collect statistical information on how your scripts are used.

Handling MobiLink server errors with .NET

When scanning the log is not sufficient, you can monitor your applications programmatically. For example, you can send messages of a certain type in an email.

You can write methods that are passed a class representing every error or warning message that is printed to the log. This may help you monitor and audit a MobiLink synchronization server.

The following code installs a Listener for all error messages and prints the information to a StreamWriter.

```
class TestLogListener {
    public TestLogListener( StreamWriter output_file )
    {
        _output_file = output_file;
    }
    public void errCallback( ServerContext sc, LogMessage lm )
    {
        string type;
        string user;
        if( lm.Type==LogMessage.MessageType.ERROR ) {
            type = "ERROR";
        } else if( lm.Type==LogMessage.MessageType.WARNING ) {
            type = "WARNING";
        } else {
            type = "INVALID TYPE!!";
        }
        if( lm.User == null ) {
            user = "null";
        } else {
            user = lm.User;
        }
        _output_file.WriteLine( "Caught msg type=" + type +
                                " user=" + user +
                                " text=" + lm.Text );
        _output_file.Flush();
    }
    StreamWriter _output_file;
}

// Two lines that registers the TestLogListener Call this code
// from anywhere that has access to the ServerContext such as
// a class constructor or synchronization script.
// ServerContext serv_context;
TestLogListener etll = new TestLogListener(log_listener_file);
serv_context.ErrorListener += new LogCallback(etll.errCallback);
```

See also

[“LogCallback delegate” on page 277](#)

ErrorListener and WarningListener in [“ServerContext interface” on page 277](#)

[“LogMessage class” on page 277](#)

[“MessageType enumeration” on page 277](#)

Debugging .NET synchronization logic

The following procedure describes one way you can debug your .NET scripts using Microsoft Visual Studio .NET.

❖ To debug .NET scripts

1. Compile your code with debugging information turned on:
 - ◆ On the csc command line, set the flag **/debug+**
or
 - ◆ Use Microsoft Visual Studio settings to set debug output

2. Load MobiLink in Microsoft Visual Studio.

For example, type the following partial command line:

```
devenv /debugexe %asany9%\win32\dbmlsrv9.exe -c ...
```

This causes dbmlsrv9.exe to appear in the Solution Explorer window.

3. Set up Microsoft Visual Studio for debugging .NET code:
 - ◆ In the Visual Studio Solution Explorer window, right-click dbmlsrv9.exe and choose Properties.
 - ◆ Change **Debugger Type** from **Auto** to **Mixed** or **Managed Only**.
4. Using Visual Studio, set break points in your .NET source code.
5. Start MobiLink from the Debug menu or using F5.
6. Perform a synchronization that causes the code with a breakpoint to be executed by MobiLink.

Writing .NET synchronization logic

Writing .NET synchronization logic is no different in complexity from writing any other .NET code. What is required from you is knowledge of MobiLink events, some knowledge of .NET, and knowledge of the MobiLink .NET API. The following sections help you write useful synchronization logic.

In this release, the row data for the upload and download streams are not passed to the .NET synchronization logic. .NET synchronization logic can be used to maintain state information, and implement logic around the upload and download events. For example, a `begin_synchronization` script written in .NET could store the MobiLink user ID in a variable. Scripts called later in the synchronization process can access this variable. Also, you can access the rows after they are in the consolidated database and before or after they are committed.

Using .NET reduces dependence on the consolidated database. Behavior is affected less by upgrading the consolidated database to a new version or switching to a different database-management system.

☞ For a complete description of the API, see [“MobiLink .NET API Reference” on page 269](#).

Class instances

The MobiLink synchronization server instantiates your classes at the connection level. When an event is reached for which you have written a non-static .NET method, the MobiLink synchronization server automatically constructs the class, if it has not already done so on the present database connection. To do so, it uses the class constructor.

For each database connection, once a class has been instantiated, the class persists until that connection is closed. Thus, the same instance may well be used for multiple consecutive synchronization sessions. Information present in public or private variables will thus persist across synchronizations that occur on the same connection unless explicitly cleared.

You can also use static classes or variables. In this case, the values are available across all connections in the same domain.

The MobiLink synchronization server automatically deletes your class instances only when the connection to the consolidated database is closed.

All methods in one script version called on the same connection *must belong to the same class*.

Transactions

The normal rules regarding transactions apply to .NET methods. The start and duration of database transactions is critical to the synchronization process. Transactions must be started and ended only by the MobiLink synchronization server. Explicitly committing or rolling back transactions on the synchronization connection within a .NET method violates the integrity of the synchronization process and can cause errors.

These rules apply only to the database connections created by the MobiLink synchronization server and, in particular, to SQL statements returned by methods. If your classes create other database connections, you are free to manage them as you please.

SQL-.NET data types

The following table shows SQL data types and the corresponding .NET data types for MobiLink script parameters.

SQL data type	Corresponding .NET data type
VARCHAR	string
CHAR	string
INTEGER	int
BINARY	byte []
TIMESTAMP	DateTime
INOUT INTEGER	ref int
INOUT VARCHAR	ref string
INOUT CHAR	ref string
INOUT BYTEARRAY	ref byte []

Constructors

The constructor of your class may have one of two possible signatures.

```
public ExampleClass(
    iAnywhere.MobiLink.Script.DBConnectionContext cc )
```

or

```
public ExampleClass()
```

The synchronization context passed to you is for the connection through which the MobiLink synchronization server is synchronizing the current user.

The `getConnection` method of the `DBConnectionContext` returns the same database connection that MobiLink is using to synchronize the present user. You can execute statements on this connection, but you must not commit or roll back the transaction. The MobiLink synchronization server manages the transactions.

The MobiLink synchronization server prefers to use constructors with the first signature. It only uses the void constructor if a constructor with the first signature is not present.

Methods

In general, you implement one method for each synchronization event. These methods must be public. If they are private, the MobiLink synchronization server cannot use them and will fail to recognize that they exist.

The names of the methods are not important, as long as the names match the names specified in the `ml_script` table in the consolidated database. In the examples included in the documentation, however, the method names are the same as those of the MobiLink events as this naming convention makes the .NET code easier to read.

The signature of your method should match the signature of the script for that event, except that you can truncate the parameter list if you do not need the values of parameters at the end of the list. Indeed, you should accept only the parameters you need, because overhead is associated with passing the parameters.

You cannot, however, overload the methods. In other words, you must provide *only one method* per class with the name specified in the `ml_script` table.

Return values

Methods called for a MobiLink upload or download must return a valid SQL language statement. The return type of these methods must be `String`. No other return types are allowed.

The return type of all other scripts must either be `string` or `void`. No other types are allowed. If the return type is a `string` and not `null`, the MobiLink synchronization server assumes that the string contains a valid SQL statement and executes this statement in the consolidated database as it would an ordinary SQL-language synchronization script. If a method ordinarily returns a `string` but does not wish to execute a SQL statement

against the database upon its return, it can return null.

User-defined start classes

You can define start classes that are loaded automatically when the server is started. The purpose of this feature is to allow you to write .NET code that executes at the time the MobiLink server starts the CLR—before the first synchronization. This means you can create connections or cache data before a user synchronization request.

You do this with the `MLStartClasses` option of the `dbmlsrv9 -sl dnet` option. For example, the following is part of a `dbmlsrv9` command line. It causes `mycl1` and `mycl2` to be loaded as start classes.

```
-sl dnet(-MLStartClasses=com.test.mycl1,com.test.mycl2)
```

Classes are loaded in the order in which they are listed. If the same class is listed more than once, more than one instance is created.

All start classes must be public and must have a public constructor that either accepts no arguments or accepts one argument of type `MobiLink.Script.ServerContext`.

The names of loaded start classes are output to the MobiLink log with the message “Loaded .NET start class: *classname*”.

☞ For more information about .NET CLR, see “-sl dnet option” [*MobiLink Synchronization Reference*, page 17].

☞ To see the start classes that are constructed at server start time, see “[GetStartClassInstances method](#)” on page 278.

Example

Following is a template start class. It starts a daemon thread that processes events and creates a database connection. (Not all start classes will need to create a thread but if a thread is spawned it should be a daemon thread.)

```
using System;
using System.IO;
using System.Threading;
using iAnywhere.MobiLink.Script;

namespace TestScripts
{
    public class MyStartClass {
        ServerContext    _sc;
        bool              _exit_loop;
        Thread            _thread;
        OdbcConnection   _conn;
```

```

public MyStartClass( ServerContext sc )
//=====
{
    // perform setup first so that an exception will
    // cause MobiLink startup to fail
    _sc          = sc;
    // create connection for use later
    _conn        = _sc.makeConnection();
    _exit_loop    = false;
    _thread       = new Thread( new ThreadStart( run ) );
    _thread.IsBackground = true;

    _thread.Start();
}

public void run()
//=====
{
    ShutdownCallback callback = new ShutdownCallback(
        shutdownPerformed );

    _sc.ShutdownListener += callback;
    // we can't throw any exceptions through run()
    try {
        handlerLoop();
        _conn.close();
        _conn = null;
    } catch( Exception e ) {
        // print some error output to the MobiLink log
        Console.Error.Write( e.ToString() );
        // we will die so we don't need to be notified of
        // shutdown
        _sc.ShutdownListener -= callback;
        // ask server to shutdown so that this fatal error will
        // be fixed
        _sc.Shutdown();
    }
    // shortly after return this thread will no longer
    // exist
    return;
}

public void shutdownPerformed( ServerContext sc )
//=====
// stop our event handler loop
{
    try {
        _exit_loop = true;
        // wait max 10 seconds for thread to die
        _thread.Join( 10*1000 );
    } catch( Exception e ) {
        // print some error output to the MobiLink log
        Console.Error.Write( e.ToString() );
    }
}
}

```



```
private void handlerLoop()  
//=====  
{  
    while( !_exit_loop ) {  
        // handle events in this loop  
        Thread.Sleep( 1*1000 );  
    }  
}  
}
```

.NET synchronization example

This example modifies an existing application to describe how to use .NET synchronization logic to handle the `authenticate_user` event. It creates a C# script for `authenticate_user` called *AuthUser.cs*. This script looks up the user's password in a table called `user_pwd_table` and authenticates the user based on that password.

First, add the table `user_pwd_table` to the database. Execute the following in Interactive SQL:

```
CREATE TABLE user_pwd_table (
    user_name  varchar(128) PRIMARY KEY NOT NULL,
    pwd        varchar(128)
)
```

Next, add a user and password to the table:

```
INSERT INTO user_pwd_table VALUES( 'user1', 'myPwd' )
```

Create a directory for your .NET assembly. For example:

```
mkdir c:\mlexample
```

Create a file called *AuthUser.cs* with the following contents:

```
using System;
using iAnywhere.MobiLink.Script;

namespace MLExample
{
    /// <summary>
    /// A simple example class the authenticates a user.
    /// </summary>
    /// <remarks>
    /// This simple example class will compare the password
    /// given for a user with the password in a table and accept
    /// or reject the authentication. We don't handle changing
    /// user password. To handle changing the password we could
    /// just update the user password table.</remarks>
    public class AuthClass
    {
        private DBConnection _conn;
```

```

/// <summary>
/// Create the instance of AuthClass for the given MobiLink
/// connection.</summary>
/// <remarks>
/// This instance will live for the duration of
/// the MobiLink connection. This means that this instance
/// will authenticate many users just as a connection will
/// handle many synchronizations.</remarks>
/// <param name="cc">The connection that owns this
/// instance.</param>
public AuthClass( DBConnectionContext cc )
{
    _conn    = cc.GetConnection();
}

/// <summary>
/// Handler for 'authenticate_user' MobiLink event.
/// </summary>
/// <remarks>
/// Handle the 'authenticate_user' event in the simplest way
/// possible. Don't handle password changes for any advanced
/// authStatus Codes.</remarks>
/// <param name="authStatus">The status for this
/// authenticate attempt.</param>
/// <param name="user">Name of the user to authenticate.
/// </param>
/// <param name="pwd">Password the user is authenticating
/// with.</param>
/// <param name="newPwd">The new password for the
/// authenticating user.</param>
public void DoAuthenticate(
    ref int authStatus,
    string user,
    string pwd,
    string newPwd )
{
    DBCommand  pwd_command = _conn.CreateCommand();
    pwd_command.CommandText = "select pwd from user_pwd_table"
                               + " where user_name = ? ";
    pwd_command.Prepare();

    // add a param for the user name that we can set later.
    DBParameter user_param = new DBParameter();
    user_param.DbType = SQLType.SQL_CHAR;
    // we need to set the size for SQL_VARCHAR
    user_param.Size    = (uint)user.Length;
    user_param.Value    = user;
    pwd_command.Parameters.Add( user_param );
}

```

```

        // fetch the password for this user.
        DBRowReader rr = pwd_command.ExecuteReader();
        object[] pwd_row = rr.NextRow();
        if( pwd_row == null ) {
            // user is unknown
            authStatus = 4000;
        } else {
            if( ((string)pwd_row[0]) == pwd ) {
                // password matched
                authStatus = 1000;
            } else {
                // password did not match
                authStatus = 4000;
            }
        }
        pwd_command.Close();
        rr.Close();
        return;
    }
}
}

```

Compile the file *AuthUser.cs*. You can do this on the command line or in Visual Studio .NET.

For example, the following command line will compile *AuthUser.cs* and generate an Assembly named *example.dll* in *c:\mlexample*. Substitute your install directory for *asany9*.

```

csc /out:c:\mlexample\example.dll /target:library /reference:\
asany9\win32\iAnywhere.MobiLink.Script.dll AuthUser.cs

```

Register .NET code for the *authenticate_user* event. The method you need to execute (*DoAuthenticate*) is in the namespace *MLEExample* and class *AuthClass*. Execute the following SQL:

```

call ml_add_dnet_connection_script( 'ex_version', 'authenticate_
user', 'MLEExample.AuthClass.DoAuthenticate' )
COMMIT

```

Next, run the MobiLink synchronization server with the following option. This option causes MobiLink to load all assemblies in *c:\myexample*:

```

-s1 dnet ( -MLAutoLoadPath=c:\mlexample )

```

Now, when a user synchronizes with the version *ex_version*, they are authenticated with the password from the table *user_pwd_table*.

MobiLink .NET API Reference

This section explains the MobiLink .NET interfaces and classes, and their associated methods, properties, and constructors. To use these classes, reference the assembly `\win32\iAnywhere.MobiLink.Script.dll` in your SQL Anywhere Studio installation directory.

This section focuses on C#, but there are equivalents in Embedded Visual Basic and C++.

DBCommand interface

public interface **DBCommand**
Member of **iAnywhere.MobiLink.Script**

Represents a SQL statement or database command. DBCommand can represent an update or query.

For example, the following C# code uses the DBCommand interface to execute two queries:

```
DBCommand stmt = conn.CreateCommand();

stmt.CommandText = "select t1a1, t1a2 from table1 ";

DBRowReader rs = stmt.ExecuteReader();
printResultSet( rs );
rs.Close();

stmt.CommandText = "select t2a1 from table2 ";

rs = stmt.ExecuteReader();
printResultSet( rs );
rs.Close();
stmt.Close();
```

The following C# example uses DBCommand to execute an update with parameters:

```

DBCommand cstmt = conn.CreateCommand();

cstmt.CommandText = "call myProc(?,?,?)";

cstmt.Prepare();

DBParameter param = new DBParameter();
param.DbType      = SQLType.SQL_CHAR;
param.Value       = "10000";
cstmt.Parameters.Add( param );

param             = new DBParameter();
param.DbType      = SQLType.SQL_INTEGER;
param.Value       = 20000;
cstmt.Parameters.Add( param );

param             = new DBParameter();
param.DbType      = SQLType.SQL_DECIMAL;
param.Precision   = 5;
param.Value       = new Decimal( 30000 );
cstmt.Parameters.Add( param );

// Execute update
DBRowReader rset = cstmt.ExecuteNonQuery();
cstmt.Close();

```

Prepare method	public void Prepare() Prepare the SQL statement stored in CommandText for execution.
ExecuteNonQuery() method	public int ExecuteNonQuery() Execute a non-query statement. Returns the number of rows in the database affected by the SQL statement.
ExecuteReader() method	public DBRowReader ExecuteReader() Execute a query statement returning the result set. Returns a DBRowReader for retrieving results returned by the SQL statement.
Close() method	public void Close() Close the current SQL statement or command.
CommandText property	public string CommandText The value is the SQL statement to be executed.
DBParameterCollection Parameters property	public DBParameterCollection Parameters Gets the iAnywhere.MobiLink.Script.DBParameterCollection for this DBCommand.

DBConnection interface

public interface **DBConnection**
Member of **iAnywhere.MobiLink.Script**

Represents a MobiLink ODBC connection.

This interface allows user-written synchronization logic to access an ODBC connection created by MobiLink.

Commit() method

public void **Commit()**

Commit the current transaction.

Rollback() method

public void **Rollback()**

Roll back the current transaction.

Close() method

public void **Close()**

Close the current connection.

CreateCommand()
method

public DBCommand **CreateCommand()**

Create a SQL statement or command on this connection. Returns the newly generated DBCommand.

DBConnectionContext interface

public interface **DBConnectionContext**
Member of **iAnywhere.MobiLink.Script**

Interface for obtaining information about the current database connection. This is passed to the constructor of classes containing scripts.

GetConnection method

public **iAnywhere.MobiLink.Script.DBConnection GetConnection()**
Member of **iAnywhere.MobiLink.Script.DBConnectionContext**

Returns the existing connection. The connection is the same connection that MobiLink uses to execute SQL scripts.

This connection must not be committed, closed or altered in any way that would affect the MobiLink server use of the connection. The connection returned is only valid for the lifetime of the underlying MobiLink connection. Do not use the connection after the end_connection event has been called for the connection.

If a server connection with full access is required, use

GetServerContext
method

ServerContext.makeConnection().

public **iAnywhere.MobiLink.Script.ServerContext.GetServerContext()**
Member of **iAnywhere.MobiLink.Script.DBConnectionContext**

Returns the ServerContext for this MobiLink server.

DBParameter class

public class **DBParameter**
Member of **iAnywhere.MobiLink.Script**

Represents a bound ODBC parameter.

DBParameter is required to execute commands with parameters. All parameters must be in place before the command is executed.

For example, the following C# code uses DBCommand to execute an update with parameters:

```
DBCommand cstmt = conn.CreateCommand();

cstmt.CommandText = "call myProc( ?,?,? )";

cstmt.Prepare();

DBParameter param = new DBParameter();
param.DbType      = SQLType.SQL_CHAR;
param.Value       = "10000";
cstmt.Parameters.Add( param );

param             = new DBParameter();
param.DbType      = SQLType.SQL_INTEGER;
param.Value       = 20000;
cstmt.Parameters.Add( param );

param             = new DBParameter();
param.DbType      = SQLType.SQL_DECIMAL;
param.Precision   = 5;
param.Value       = new Decimal( 30000 );
cstmt.Parameters.Add( param );

// Execute update
DBRowReader rset = cstmt.ExecuteNonQuery();
cstmt.Close();
```

dbType property

public **SQLTYPE dbType**

The value is the SQLType of this parameter.

Default: SQLType.SQL_TYPE_NULL.

Direction property

public **System.Data.ParameterDirection Direction**

	<p>The value is the Input/Output direction of this parameter.</p> <p>Default: ParameterDirection.Input.</p>
IsNullable property	<p>public bool IsNullable</p> <p>The value Indicates whether this parameter can be NULL.</p> <p>Default: false.</p>
ParameterName property	<p>public string ParameterName</p> <p>The value is the name of this parameter.</p> <p>Default: null.</p>
Precision property	<p>public uint Precision</p> <p>The value is the decimal precision of this parameter. Only used for SQLType.SQL_NUMERIC and SQLType.SQL_DECIMAL parameters.</p> <p>Default: 0.</p>
Scale property	<p>public short Scale</p> <p>The value is the resolvable digits of this parameter. Only used for SQLType.SQL_NUMERIC and SQLType.SQL_DECIMAL parameters.</p> <p>Default: 0.</p>
Size property	<p>public uint Size</p> <p>The value is the size in bytes of this parameter.</p> <p>Default: Inferred from DbType.</p>
Value property	<p>public object Value</p> <p>The value is the value of this parameter.</p> <p>Default: null.</p>

DBParameterCollection class

	<p>public class DBParameterCollection</p> <p>inherits from IDataParameterCollection, IList, ICollection, IEnumerable</p> <p>Member of iAnywhere.MobiLink.Script</p> <p>Collection of DBParameters. When DBCommand creates a DBParamterCollection it is empty and must be filled with appropriate parameters before the DBCommand executes.</p>
DBParameterCollection() method	<p>public DBParameterCollection()</p>

	Creates an empty list of DBParameters.
Contains(string parameterName) method	<p>public bool Contains(string parameterName)</p> <p>Returns true if the collection contains a parameter with the specified name. Takes one parameter, <i>parameterName</i>, which is the name of the parameter.</p>
IndexOf(string parameterName) method	<p>public int IndexOf(string parameterName)</p> <p>Returns index of the parameter, or -1 if there is no parameter with the given name. Takes one parameter, <i>parameterName</i>, which is the name of the parameter.</p>
RemoveAt(string parameterName) method	<p>public void RemoveAt(string parameterName)</p> <p>Removes the parameter with the given name from the collection. Takes one parameter, <i>parameterName</i>, which is the name of the parameter.</p>
Add(object value) method	<p>public int Add(object value) method</p> <p>Adds the given parameter to the collection. Takes one parameter, <i>value</i>, which is the iAnywhere.MobiLink.Script.DBParameter to add to the collection. Returns the index of the added parameter in the collection.</p>
Clear() method	<p>public void Clear()</p> <p>Removes all parameters from the collection.</p>
Contains(object value) method	<p>public bool Contains(object value) method</p> <p>Returns true if this collection contains the given iAnywhere.MobiLink.Script.DBParameter. Takes one parameter, <i>value</i>, which is the iAnywhere.MobiLink.Script.DBParameter.</p>
IndexOf(object value) method	<p>public int IndexOf(object value)</p> <p>Returns the index of the given iAnywhere.MobiLink.Script.DBParameter in the collection. Takes one parameter, <i>value</i>, which is the iAnywhere.MobiLink.Script.DBParameter.</p>
Insert(int index, object value) method	<p>public void Insert(int index, object value)</p> <p>Inserts the given iAnywhere.MobiLink.Script.DBParameter into the collection at the specified index. Takes two parameters: <i>value</i>, which is the iAnywhere.MobiLink.Script.DBParameter; and <i>index</i>, which is the index to insert at.</p>
Remove(object value) method	<p>public void Remove(object value)</p> <p>Removes the given iAnywhere.MobiLink.Script.DBParameter from the</p>

	collection. Takes one parameter, <i>value</i> , which is the <code>iAnywhere.MobiLink.Script.DBParameter</code> .
RemoveAt(int index) method	<p>public int RemoveAt(int index)</p> <p>Removes the <code>iAnywhere.MobiLink.Script.DBParameter</code> at the given index in the collection. Takes one parameter, <i>index</i>, which is the index of the <code>iAnywhere.MobiLink.Script.DBParameter</code>.</p>
CopyTo(Array array, int index) method	<p>public void CopyTo(Array array, int index)</p> <p>Copies the contents of the collection into the given array starting at the specified index. Takes two parameters: <i>array</i>, which is the array to copy the contents of the collection into; and <i>index</i>, which is the index in the array to begin copying the contents of the collection into.</p>
GetEnumerator() method	<p>public IEnumerator GetEnumerator()</p> <p>Returns an enumerator for the collection.</p>
IsFixedSize property	<p>public bool IsFixedSize</p> <p>Returns false.</p>
IsReadOnly property	<p>public bool IsReadOnly</p> <p>Returns false.</p>
Count property	<p>public int Count</p> <p>The number of parameters in the collection.</p>
IsSynchronized property	<p>public bool IsSynchronized</p> <p>Returns false.</p>
SyncRoot property	<p>public object SyncRoot</p> <p>Object that can be used to synchronize the collection.</p>
this[string parameterName] property	<p>public object this[string parameterName]</p> <p>Gets or sets the <code>iAnywhere.MobiLink.Script.DBParameter</code> with the given name in the collection. Takes one parameter, <i>parameterName</i>, which is the name of the <code>iAnywhere.MobiLink.Script.DBParameter</code> to get or set.</p>
this[int index] property	<p>public object this[int index]</p> <p>Gets or sets the <code>iAnywhere.MobiLink.Script.DBParameter</code> at the given index in the collection. Takes one parameter, <i>index</i>, which is the index of the <code>iAnywhere.MobiLink.Script.DBParameter</code> to get or set.</p>

DBRowReader interface

public interface **DBRowReader**
Member of **iAnywhere.MobiLink.Script**

Represents a set of rows being read from a database. Executing the method `DBCommand.executeReader()` creates a `DBRowReader`.

The following example is a C# code fragment. It calls a function with the rows in the result set represented by the given `DBRowReader`.

```
DBCommand stmt = conn.CreateCommand();

stmt.CommandText = "select intCol, strCol from table1 ";

DBRowReader rs = stmt.ExecuteReader();
object[] values = rset.NextRow();

while( values != null ) {
    handleRow( (int)values[0], (String)values[1] );
    values = rset.NextRow();
}
rset.Close();
stmt.Close();
```

`NextRow()` method

public object[] **NextRow()**

Retrieves and returns the next row of values in the result set. If there are no more rows in the result set, it returns NULL.

☞ See [“SQLType enumeration” on page 280](#).

`Close()` method

public void **Close()**

Cleans up resources used by this `MLDBRowReader`. After `Close()` is called, this `MLDBRowReader` cannot be used again.

`ColumnNames` property

public string[] **ColumnNames**

Gets the names of all columns in the result set. The value is an array of strings corresponding to the column names in the result set.

`ColumnTypes` property

public SQLType[] **ColumnTypes**

Gets the types of all columns in the result set. The value is an array of `SQLTypes` corresponding to the column types in the result set.

LogCallback delegate

```
public delegate void LogCallback(  
    ServerContext sc  
    LogMessage message  
)  
Member of iAnywhere.MobiLink.Script
```

Called when the MobiLink synchronization server prints a message.

LogMessage class

```
public class LogMessage : iAnywhere.MobiLink.Script.LogMessage  
Member of iAnywhere.MobiLink.Script
```

Contains information about a message printed to the log.

Type property

```
public LogMessage.MessageType Type
```

The type of the log message that this instance represents.

User property

```
public string User
```

The user for which this message is being logged. It may be null.

Text property

```
public string Text
```

The main text of the message.

MessageType enumeration

```
public enum MessageType  
Member of iAnywhere.MobiLink.Script.LogMessage
```

Enumeration of the possible types of LogMessage.

ERROR field

```
public ERROR
```

A log error message.

WARNING field

```
public WARNING
```

A log warning message.

ServerContext interface

```
public interface ServerContext  
Member of iAnywhere.MobiLink.Script
```

An instantiation of all the context that is present for the duration of the MobiLink server. This context can be held as static data and used in a background thread. It is valid for the duration of the .NET CLR invoked by MobiLink.

GetStartClassInstances
method

public **object[] GetStartClassInstances()**
Member of **iAnywhere.MobiLink.Script.ServerContext**

Gets an array of the start classes that were constructed at server start time. The array length is zero if there are no start classes.

☞ For more information about user-defined start classes, see [“User-defined start classes” on page 263](#).

Following is an example of getStartClassInstances():

```
void FindStartClass( ServerContext sc, string name )
{
    object[] startClasses = sc.GetStartClassInstances();

    foreach( object obj in startClasses ) {
        if( obj is MyClass ) {
            // Execute some code.....
        }
    }
}
```

LogCallback
ErrorListener event

This event is triggered when the MobiLink synchronization server prints an error.

LogCallback
WarningListener event

This event is triggered when the MobiLink synchronization server prints a warning.

MakeConnection method

public **iAnywhere.MobiLink.Script.DBConnection makeConnection()**
Member of **iAnywhere.MobiLink.Script.ServerContext**

Opens and returns a new server connection. To access the server context, use DBConnectionContext.getServerContext on the synchronization context for the current connection.

ShutDown method

public void **Shutdown()**
Member of **iAnywhere.MobiLink.Script.ServerContext**

Forces the server to shut down.

ShutdownListener
method

public event iAnywhere.MobiLink.Script.ShutdownCallback **ShutdownListener(**
 iAnywhere.MobiLink.Script.ServerContext sc)
Member of **iAnywhere.MobiLink.Script.ServerContext**

This event is triggered on shutdown. The following code is an example of how to use this event:

```

ShutdownCallback callback = new ShutdownCallback(
    shutdownHandler );
_sc.ShutdownListener += callback;

public void shutdownHandler( ServerContext sc )
//=====
{
    _test_out_file.WriteLine( "shutdownPerformed" );
}

```

ServerException class

public class **ServerException** : **iAnywhere.MobiLink.Script.ScriptExecutionException**
 Member of **iAnywhere.MobiLink.Script**

Used to signal MobiLink that an error has occurred with the server and it should shut down immediately.

ServerException
 constructors

public **ServerException()**
 Member of **iAnywhere.MobiLink.Script.ServerException**

Constructs a ServerException with no detail message.

public **ServerException(string message)**
 Member of **iAnywhere.MobiLink.Script.ServerException**

Creates a new ServerException with the given message. The parameter *message* is the message for this ServerException.

public **ServerException(string message, SystemException ie)**
 Member of **iAnywhere.MobiLink.Script.ServerException**

Creates a new ServerException with the given message and containing the given inner exception that caused this one. There are two parameters: *message*, which is the message for this ServerException, and *ie*, which is the exception that caused this ServerException.

ShutdownCallback delegate

public sealed delegate **ShutdownCallback** : **System.MulticastDelegate**
 Member of **iAnywhere.MobiLink.Script**

Called when the MobiLink synchronization server is shutting down. Implementations of this delegate can be registered with the `ServerContext.ShutdownListener` event to be called when the MobiLink server shuts down.

SQLType enumeration

	public enum SQLType Member of iAnywhere.MobiLink.Script Enumeration of all possible ODBC data types.
SQL_TYPE_NULL field	public SQL_TYPE_NULL Null data type.
SQL_UNKNOWN_TYPE field	public SQL_UNKNOWN_TYPE Unknown data type.
SQL_CHAR field	public SQL_CHAR UTF-8 character array of a set size. Has .NET type String.
SQL_NUMERIC field	public SQL_NUMERIC Numeric value of set size and precision. Has .NET type Decimal.
SQL_DECIMAL field	public SQL_DECIMAL Decimal number of set size and precision. Has .NET type Decimal.
SQL_INTEGER field	public SQL_INTEGER 32-bit integer. Has .NET type Int32.
SQL_SMALLINT field	public SQL_SMALLINT 16-bit integer. Has .NET type Int16.
SQL_FLOAT field	public SQL_FLOAT Floating point number with ODBC driver defined precision. Has .NET type Double.
SQL_REAL field	public SQL_REAL Single precision floating-point number. Has .NET type Single.
SQL_DOUBLE field	public SQL_DOUBLE Double precision floating point number. Has .NET type Double.
SQL_DATE field	public SQL_DATE A date. Has .NET type DateTime.

SQL_DATETIME field	public SQL_DATETIME A date and time. Has .NET type DateTime.
SQL_TIME field	public SQL_TIME A time. Has .NET type DateTime.
SQL_INTERVAL field	public SQL_INTERVAL An interval of time. Has .NET type TimeSpan.
SQL_TIMESTAMP field	public SQL_TIMESTAMP A time stamp. Has .NET type DateTime.
SQL_VARCHAR field	public SQL_VARCHAR A null terminated UTF-8 string with a user set maximum length. Has .NET type String.
SQL_TYPE_DATE field	public SQL_TYPE_DATE A date. Has .NET type DateTime.
SQL_TYPE_TIME field	public SQL_TYPE_TIME A time. Has .NET type DateTime.
SQL_TYPE_TIMESTAMP field	public SQL_TYPE_TIMESTAMP A timestamp. Has .NET type DateTime.
SQL_DEFAULT field	public SQL_DEFAULT A default type. Has no type.
SQL_ARD_TYPE field	public SQL_ARD_TYPE An ARD object. Has no type.
SQL_BIT field	public SQL_BIT A single bit. Has .NET type Boolean.
SQL_TINYINT field	public SQL_TINYINT An 8-bit integer. Has .NET type SByte.
SQL_BIGINT field	public SQL_BIGINT A 64-bit integer. Has .NET type Int64.

SQL_LONGVARBINARY field	public SQL_LONGVARBINARY Variable length binary data with a driver dependent maximum length. Has .NET type byte[].
SQL_VARBINARY field	public SQL_VARBINARY Variable length binary data with a user specified maximum length. Has .NET type byte[].
SQL_BINARY field	public SQL_BINARY Fixed length binary data. Has .NET type byte[].
SQL_LONGVARCHAR field	public SQL_LONGVARCHAR A null-terminated UTF-8 string with a driver-dependent maximum length. Has .NET type String.
SQL_GUID field	public SQL_GUID A Global Unique ID (also called a UUID). Has .NET type Guid.
SQL_WCHAR field	public SQL_WCHAR Unicode character array of fixed size. Has .NET type String.
SQL_WVARCHAR field	public SQL_WVARCHAR Null-terminated Unicode string of user-defined maximum length. Has .NET type String.
SQL_WLONGVARCHAR field	public SQL_WLONGVARCHAR Null-terminated Unicode string of driver-dependent maximum length. Has .NET type String.

SynchronizationException class

	public class SynchronizationException : iAnywhere.MobiLink.Script. ScriptExecutionException Member of iAnywhere.MobiLink.Script Used to signal that a synchronization exception has occurred and that the current synchronization should be rolled back and restarted.
SynchronizationException constructors	public SynchronizationException() Member of iAnywhere.MobiLink.Script.SynchronizationException Constructs a SynchronizationException with no details.

public **SynchronizationException**(string *message*)

Member of **iAnywhere.MobiLink.Script.SynchronizationException**

Creates a new SynchronizationException with the given message. The parameter *message* is the message for this ServerException.

public **SynchronizationException**(string *message*, **SystemException** *ie*)

Member of **iAnywhere.MobiLink.Script.SynchronizationException**

Creates a new SynchronizationException with the given message and containing the given inner exception that caused this one. There are two parameters: *message*, which is the message for this ServerException, and *ie*, which is the exception that caused this ServerException.

CHAPTER 12

MobiLink Performance

About this chapter

This chapter provides information that can help you improve the performance of your MobiLink synchronization.

For more information about MobiLink performance, see the *MobiLink Performance* whitepaper at <http://my.sybase.com/detail?id=1009664>.

Contents

Topic:	page
Performance tips	286
Key factors influencing MobiLink performance	290
Monitoring MobiLink performance	295

Performance tips

Following are some suggestions to help you get the best performance out of MobiLink.

- ◆ **Test** Before deploying, perform volume testing using the same hardware and network that you plan to use for production. Use this time to experiment with the following performance tips.

- ◆ **Avoid contention** Avoid contention in your synchronization scripts. Another way of putting this is that you should maximize concurrency.

For example, suppose a `begin_download` script increments a column in a table to count the total number of downloads. If multiple users synchronize at the same time, this script would effectively serialize their downloads. The same counter would be better in the `begin_synchronization` or `end_synchronization` script because these scripts are called just before a commit.

☞ For more information about contention, see [“Contention” on page 291](#).

☞ For information on the transaction structure of synchronization, see [“Transactions in the synchronization process” on page 25](#).

- ◆ **Use an optimal number of worker threads** Use the MobiLink `-w` option to set the number of MobiLink worker threads to the smallest number that gives you optimum throughput. You will need to experiment to find the best number for your situation.

A larger number of worker threads can improve throughput by allowing more synchronizations to occur at the same time.

Keeping the number of worker threads small reduces the chance of contention in the consolidated database, the number of connections to the consolidated database, and the memory required for optimal caching.

For example, in tests with fast clients, it was discovered that approximately five worker threads gave optimum throughput. For slower clients, more worker threads were needed to maximize download throughput, and the best upload throughput was obtained by limiting the number that can simultaneously upload, via the `-wu` option. In tests with extremely slow clients, the best throughput for both uploads and downloads was obtained with hundreds of worker threads with only five allowed to upload simultaneously. Note that these numbers are from a specific set of tests. Every deployment has different characteristics, and you must test to determine the optimal values for `-w` and `-wu`.

☞ For more information about worker threads, see [“Number of worker threads” on page 292](#).

☞ For more information, see “-w option” [*MobiLink Synchronization Reference*, page 22] and “-wu option” [*MobiLink Synchronization Reference*, page 23].

- ◆ **Enable the client-side download buffer for ASA clients** For Adaptive Server Anywhere clients, a download buffer allows a MobiLink worker thread to transmit the download without waiting for the client to apply the download. The download buffer is enabled by default. However, the download buffer cannot be used if download acknowledgement is enabled (see next bullet).

☞ For more information about setting the download buffer size, see the “DownloadBufferSize (dbs) extended option” [*MobiLink Synchronization Reference*, page 48].

- ◆ **Disable download acknowledgement for ASA clients** Eliminating the optional download acknowledgement can free up MobiLink worker threads that are waiting for confirmation of successful download from the client, which also frees up the connection that the worker thread is using. It also makes it possible for MobiLink synchronization server to buffer the downloads.

You can also disable the download acknowledgement for UltraLite clients, but there is little performance improvement because UltraLite clients do not buffer downloads.

☞ For more information about download acknowledgements, see the “SendDownloadACK (sa) extended option” [*MobiLink Synchronization Reference*, page 62].

- ◆ **Set the upload cache size** To avoid the situation where the upload cache overflows to disk, set the upload cache size to be larger than the size of your largest upload stream times the number of worker threads. You set the upload cache size with the dbmlsrv9 -u option.

☞ For more information, see “-u option” [*MobiLink Synchronization Reference*, page 20].

- ◆ **Set the download cache size** To avoid the situation where the download buffer overflows to disk, set the download cache size to be larger than the size of your largest download times the number of worker threads. You set the download cache size with the dbmlsrv9 -d option.

☞ For more information about setting the memory allocated to the download buffer, see “-d option” [*MobiLink Synchronization Reference*, page 11].

- ◆ **Set the BLOB cache size** If your rows have data of type LONG VARCHAR or LONG BINARY, you can avoid having the BLOB cache

access disk if you set the BLOB cache size to be larger than twice the largest BLOB data in a row times the number of worker threads. You set the BLOB cache size with the `dbmlsrv9 -bc` option.

☞ For more information, see “-bc option” [*MobiLink Synchronization Reference*, page 9].

- ◆ **Set maximum number of database connections** Set the maximum number of MobiLink database connections to be your typical number of synchronization script versions times the number of MobiLink worker threads, plus one. This reduces the need for MobiLink to close and create database connections. You set the maximum number of connections with the `dbmlsrv9 -cn` option.

☞ For more information, see “[MobiLink database connections](#)” on [page 293](#) and “-cn option” [*MobiLink Synchronization Reference*, page 11].

- ◆ **Have sufficient physical memory** Ensure that the computer running MobiLink has enough physical memory to accommodate the upload, download and BLOB caches in addition to its other memory requirements.
- ◆ **Use sufficient processing power** Dedicate enough processing power to MobiLink so that the MobiLink server processing is not a bottleneck. In tests with an Adaptive Server Anywhere consolidated database, MobiLink required a third to a half of the processing required by Adaptive Server Anywhere when both were stressed and executing on the same computer.
- ◆ **Use minimum logging verbosity** Use the minimum logging verbosity that is compatible with your business needs. By default, verbose logging is off, and MobiLink does not write its log to disk. You can control logging verbosity with the `-v` option, and enable logging to a file with the `-o` or `-ot` options.

As an alternative to verbose log files, you can monitor your synchronizations with the MobiLink Monitor. The Monitor does not need to be on the same computer as the MobiLink synchronization server, and a Monitor connection has negligible effect on MobiLink server performance. For more information, see “[MobiLink Monitor](#)” on [page 297](#).

- ◆ **Java or .NET vs. SQL synchronization logic** No significant throughput difference has been found between using Java or .NET synchronization logic vs. SQL synchronization logic. However, Java and .NET synchronization logic have some extra overhead per synchronization and require more memory.

In addition, SQL synchronization logic is executed on the computer that runs the consolidated database, while Java or .NET synchronization logic is executed on the computer that runs the MobiLink server. Thus, Java or .NET synchronization logic may be desirable if your consolidated database is heavily loaded.

- ◆ **Priority synchronization** If you have some tables that you need to synchronize more frequently than others, create a separate publication and subscription for them. You can synchronize this priority publication more frequently than other publications, and synchronize other publications at off-peak times.
- ◆ **Download only the rows you need** Take care to download only the rows that are required. It is easier to write synchronization scripts that download all rows upon each synchronization, but downloading unneeded rows affects synchronization performance.
- ◆ **Optimize script execution** The performance of your scripts in the consolidated database is an important factor. It may help to create indexes on your tables so that the upload and download cursor scripts can efficiently locate the required rows. However, too many indexes may slow uploads.
- ◆ **For large uploads from ASA clients, estimate the number of rows** You can significantly improve the speed of uploading a large number of rows by providing dbmlsync with an estimate of the number of rows that will be uploaded. You do this with the dbmlsync -urc option.

☞ For more information, see “-urc option” [*MobiLink Synchronization Reference*, page 80].

Key factors influencing MobiLink performance

The overall performance of any system, including throughput for MobiLink synchronization, is usually limited by a bottleneck at one point in the system. For MobiLink synchronization, the following might be the bottlenecks limiting synchronization throughput:

- ◆ **The performance of the consolidated database** Of particular importance for MobiLink is the speed at which it can execute the MobiLink scripts. Multiple worker threads might execute scripts simultaneously, so for best throughput you need to avoid database contention in your synchronization scripts.
- ◆ **The bandwidth for MobiLink to consolidated communication** This is unlikely to be a bottleneck if both MobiLink and the consolidated database are running on the same computer, or if they are on separate computers connected by a high-speed network.
- ◆ **The speed of the computer running MobiLink** If the processing power of the computer running MobiLink is slow, or if it does not have sufficient memory for the MobiLink worker threads and buffers, then MobiLink execution speed could be a synchronization bottleneck. The MobiLink server's performance depends little on disk speed as long as the buffers and worker threads fit in physical memory.
- ◆ **The number of MobiLink worker threads** A smaller number of threads will involve fewer database connections, less chance of contention in the consolidated database and less operating system overhead. However, too small a number may leave clients waiting for a free worker thread, or have fewer connections to the consolidated database than it can overlap efficiently.
- ◆ **The bandwidth for client-to-MobiLink communications** For slow connections, such as those over dial-up or wide-area wireless networks, the network may cause clients and MobiLink worker threads to wait for data to be transferred.
- ◆ **The client processing speed** Slow client processing speed is more likely to be a bottleneck in downloads than uploads, since downloads involve more client processing as rows and indexes are written.

Tuning MobiLink for performance

The key to achieving optimal MobiLink synchronization throughput is to have multiple synchronizations occurring simultaneously and executing efficiently. To enable multiple simultaneous synchronizations, MobiLink

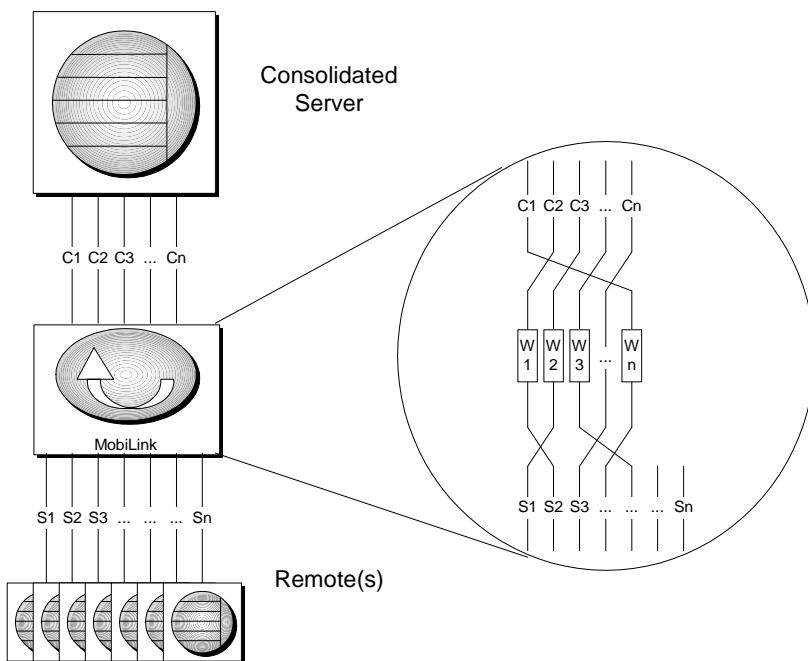
assigns a worker thread to each synchronization. A worker thread receives the changes uploaded from the client and applies them to the consolidated database. It then fetches the changes from the consolidated database, and downloads them to the client. Each worker thread uses a single connection to the consolidated database for applying and fetching changes, using your synchronization scripts.

Contention

The most important factor is to avoid database contention in your synchronization scripts. Just as with any other multi-client use of a database, you want to minimize database contention when clients are simultaneously accessing a database. Database rows that must be modified by each synchronization can increase contention. For example, if your scripts increment a counter, then updating that counter can be a bottleneck.

The figure below shows the following:

- ◆ a pool of connections to the consolidated database, shown as C1 to Cn
- ◆ a number of synchronization requests, shown as S1 to Sn
- ◆ MobiLink worker threads, shown as W1 to Wn



If there are more synchronization requests than worker threads, the excess requests are queued until a worker thread becomes available after completing a synchronization. You can control the number of worker threads

Number of worker threads

and connections, but MobiLink will always ensure that there is at least one connection per worker thread. If there are more connections than worker threads, the excess connections will be idle. Excess connections may be useful with multiple script versions, as discussed below.

Other than contention in your synchronization scripts, the most important factor for synchronization throughput is the number of worker threads. The number of worker threads controls how many synchronizations can proceed simultaneously.

Testing is vital to determine the optimum number of worker threads.

Increasing the number of worker threads allows more overlapping synchronizations, and increased throughput, but it will also increase resource and database contention between the overlapping synchronizations, and increase the time for individual synchronizations. As the number of worker threads is increased, the benefit of more simultaneous synchronizations becomes outweighed by the cost of longer individual synchronizations, and adding more worker threads decreases throughput. Experimentation is required to determine the optimal number of worker threads for your situation, but the following may help to guide you.

For uploads, performance testing shows that the best throughput happens with a relatively small number of worker threads: in most cases, three to ten worker threads. Variation depends on factors like the type of consolidated database, data volume, database schema, the complexity of the synchronization scripts, and the hardware used. The bottleneck is usually due to contention between worker threads executing the SQL of your upload scripts at the same time in the consolidated database.

For downloads, the optimum number of worker threads depends on the client to MobiLink bandwidth and the processing speed of clients. For slower clients, more worker threads are needed to get optimal download performance. This is because downloads involve more client processing and less consolidated database processing than uploads.

For Adaptive Server Anywhere clients, eliminating the download acknowledgement (and not disabling the optional download buffering) can reduce the optimal number of worker threads for download, because worker threads do not have to wait for clients to apply downloads. There is little effect for UltraLite clients since UltraLite clients apply the download as it is received, without buffering.

☞ For more information on disabling the download acknowledgement, see the “SendDownloadACK (sa) extended option” [*MobiLink Synchronization Reference*, page 62].

To get both the best download throughput and the best upload throughput, MobiLink provides two options. You can specify a total number of worker threads to optimize downloads. You can also limit the number that can simultaneously apply uploads to optimize upload throughput.

The `-w` option controls the total number of worker threads. The default is five.

The `-wu` option limits the number of worker threads that can simultaneously apply uploads to the consolidated database. By default, all worker threads can apply uploads simultaneously, but that can cause severe contention in the consolidated database. The `-wu` option lets you reduce that contention while still having a larger number of worker threads to optimize downloads and receive uploads. The `-wu` option only has an effect if the number is less than the total number of worker threads.

☞ For more information, see “`-w` option” [*MobiLink Synchronization Reference*, page 22] and “`-wu` option” [*MobiLink Synchronization Reference*, page 23].

MobiLink database connections

MobiLink creates a database connection for each worker thread. You can use the `-cn` option to specify that MobiLink create a larger pool of database connections, but any excess connections will be idle unless MobiLink needs to close a connection or use a different script version.

There are two cases where MobiLink will close a database connection and open a new one. The first case is if an error occurs. The second case is if the client requests a synchronization script version, and none of the available connections have already used that synchronization version.

Note

Each database connection is associated with a script version. To change the version, the connection must be closed and reopened.

If you have more than one synchronization version, you may want to set the maximum number of pooled connections to be larger than the number of worker threads, which is the default number. Then MobiLink will not need to close and open a new database connection each time a different synchronization version is requested.

If you routinely use more than one script version, you can reduce the need for MobiLink to close and open connections by increasing the number of connections. You can eliminate the need completely if the number of connections is the number of worker threads times the number of versions.

An example of tuning MobiLink for two script versions is given in the command line below:

```
dbmlsrv9 -c "dsn=ASA 9.0 Sample" -w 5 -cn 10
```

Since the maximum usable number of database connections is the number of script versions times the number of worker threads plus one, you can set `-cn` to 10 to ensure that database connections are not closed and opened to accommodate synchronization versions.

An example of tuning MobiLink for three script versions is:

```
dbmlsrv9 -c "dsn=ASA 9.0 Sample" -w 7 -cn 21
```

☞ For more information on setting the number of connections for any number of script versions, see “`-cn` option” [*MobiLink Synchronization Reference*, page 11].

Monitoring MobiLink performance

There are a variety of tools available to help you monitor the performance of your synchronizations.

The MobiLink Monitor is a graphical tool for monitoring synchronizations. It allows you to see the time taken by every aspect of the synchronization, sorted by MobiLink user or by worker thread.

☞ For more information, see [“MobiLink Monitor” on page 297](#).

In addition, there are a number of MobiLink scripts that are available for monitoring synchronizations. These scripts allow you to use performance statistics in your business logic. You may, for example, want to store the performance information for future analysis, or alert a DBA if a synchronization takes too long. For more information, see

- ◆ “download_statistics connection event” [*MobiLink Synchronization Reference*, page 139]
- ◆ “download_statistics table event” [*MobiLink Synchronization Reference*, page 142]
- ◆ “synchronization_statistics connection event” [*MobiLink Synchronization Reference*, page 202]
- ◆ “synchronization_statistics table event” [*MobiLink Synchronization Reference*, page 205]
- ◆ “time_statistics connection event” [*MobiLink Synchronization Reference*, page 207]
- ◆ “time_statistics table event” [*MobiLink Synchronization Reference*, page 209]
- ◆ “upload_statistics connection event” [*MobiLink Synchronization Reference*, page 224]
- ◆ “upload_statistics table event” [*MobiLink Synchronization Reference*, page 227]

CHAPTER 13

MobiLink Monitor

About this chapter

The MobiLink Monitor is a tool for monitoring MobiLink synchronizations.

Contents

Topic:	page
Introduction	298
Starting the MobiLink Monitor	299
Using the MobiLink Monitor	302
Saving Monitor data	307
Customizing your statistics	308
MobiLink statistical properties	310

Introduction

The MobiLink Monitor is a MobiLink administration tool that provides you with detailed information about the performance of your synchronizations.

When you start the Monitor and connect it to a MobiLink synchronization server, the Monitor begins to collect statistical information about all synchronizations that occur in that monitoring session. The Monitor continues to collect data until you disconnect it or shut down the MobiLink server.

You can view the data in tabular or graphical form in the Monitor interface. You can also save the data in binary format for viewing with the Monitor later, or in .csv format to open in another tool, such as Microsoft Excel.

Monitor output allows you to see a wide variety of information about your synchronizations. For example, you can quickly identify synchronizations that result in errors, or that meet other criteria that you specify. You can identify possible contention in synchronization scripts by checking to see if synchronizations of differing durations have phases that end around the same time (because synchronizations are waiting for a previous phase to finish before they can continue).

The MobiLink Monitor can be used routinely in development and production, because monitoring does not degrade performance, particularly when the Monitor is run on a different computer from the MobiLink synchronization server.

Starting the MobiLink Monitor

If synchronization is already occurring when the MobiLink Monitor is started, the Monitor must wait until a worker thread is free before it can start monitoring. Therefore, you may want to start the Monitor before starting synchronizations. Once the Monitor is running it does not use a MobiLink worker thread.

You can have one instance of the Monitor running for each MobiLink synchronization server.

❖ To start monitoring data

1. From the Start menu, choose Programs ► SQL Anywhere 9 ► MobiLink ► MobiLink Monitor.

Alternatively, you can type **dbmlmon** at a command prompt. For details, see below.

2. Start your consolidated database and MobiLink synchronization server, if they are not already running.
3. In the MobiLink Monitor, choose Monitor ► Connect to MobiLink Server.

The Connect to MobiLink Server dialog appears.

A Monitor connection starts like a synchronization connection to the MobiLink synchronization server. For example, if you started the MobiLink server with `-zu+` then it doesn't matter what user ID you use here. For all MobiLink Monitor sessions, the script version is set to `for_ML_Monitor_only`.

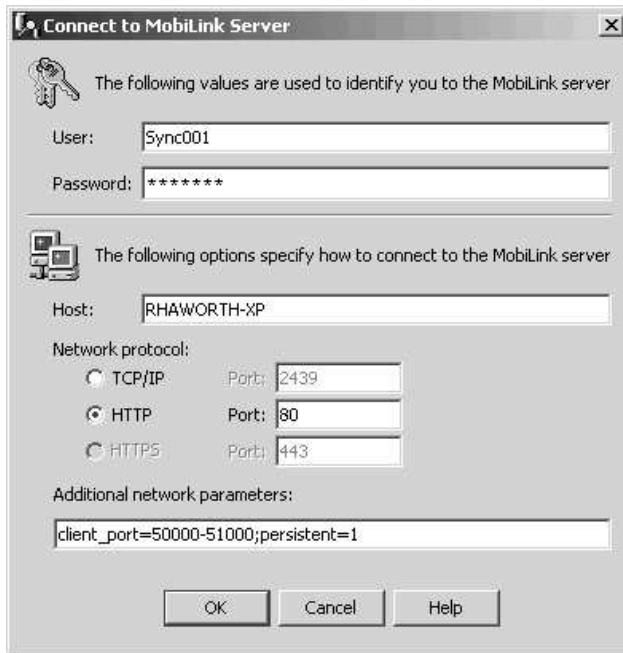
The Connect to MobiLink Server dialog should be completed as follows:

- ◆ **Host** is the computer where the MobiLink synchronization server is running. By default, it is the computer where the Monitor is running.
- ◆ **Network Protocol** should be set to the same protocol and port as the MobiLink synchronization server is using for synchronization requests.
- ◆ **Additional Network Parameters** allows you to set optional parameters. You can set the following parameters, separated by semi-colon if you need to specify multiple parameters:
 - **buffer_size=number** (HTTP and HTTPS only)
 - **client_port=nnnn**
 - **client_port=nnnn-mmmmm**
 - **persistent={0|1}**
 - **proxy_host=proxy_hostname** (HTTP and HTTPS only)

- **proxy_port**=*proxy_portnumber* (HTTP and HTTPS only)
- **url_suffix**=*suffix* (HTTP and HTTPS only)
- **version**=*versionnumber* (HTTP and HTTPS only)

☞ For more information about these network parameters, see “Stream parameters reference” [*UltraLite Database User’s Guide*, page 179].

In the following example, the outgoing port range is restricted to 50000-51000, and a persistent HTTP connection is used.



4. Start synchronizing.

The data appears in the Monitor as it is collected.

Starting dbmlmon on the command line

You can also start the MobiLink Monitor on the command line, using the following syntax:

```
dbmlmon [ connect-options | inputfile.{ mlm | csv } | -? ]
```

where:

connect-options can be one or more of the following:

-u *ml_username*

-p *password*

-x { *tcpip* | *http* | *https* } [(*keyword=value*;...)]

-o *outputfile*.{ **mlm** | **csv** }

-? You can type **dbmlmon -?** to view the dbmlmon syntax.

❖ To stop the MobiLink Monitor

1. In the Monitor, choose Monitor ► Disconnect from MobiLink Server.
This stops the collection of data.

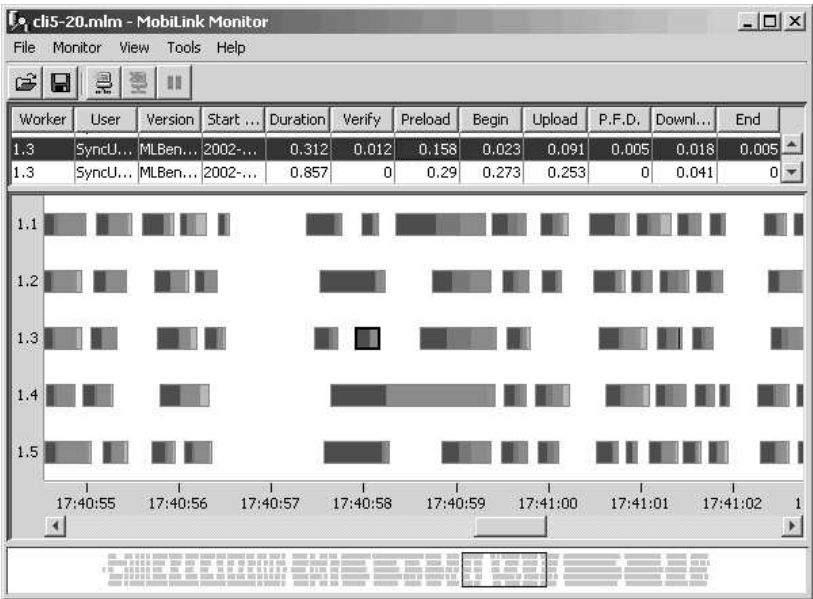
You can also stop collecting data by shutting down the MobiLink synchronization server or closing the Monitor.

Before closing the Monitor, you can save the data for the session. For more information, see [“Saving Monitor data” on page 307](#).

2. When you are ready to close the Monitor, choose File ► Close.

Using the MobiLink Monitor

Following is an example of the MobiLink Monitor when synchronization data has been collected:



The Monitor has three panes:

- ◆ **Details Table** is the top pane. It is a spreadsheet that shows the total time taken by each synchronization, with a breakdown showing the amount of time taken by each part of the synchronization.
- ◆ **Chart** is the middle pane. It provides a graphical representation of the data. The scale at the bottom of this pane represents time. You can select the data that is displayed in the Chart by drawing a box around data in the Overview pane; or by choosing View ► Go To.

In the screen shot above, the cursor is hovering over the time scale, and so a box is apparent that shows the complete date and time for the position of the cursor.

- ◆ **Overview** is the bottom pane. It shows an overview of all the data. To choose data to see in the Chart, click in the Overview and draw a box. The Chart will show everything that is located in the box.

In addition, there is an Options dialog that you can use to customize the data, and properties dialogs for viewing more detailed information. All of these panes and dialogs are described in detail, below.

Details Table pane

The Details Table provides information about how long each part of the synchronization took. All times are measured by the MobiLink synchronization server. Some times may be non-zero even when you do not have the corresponding script defined.

The Details Table has the following columns:

- ◆ **Worker** Identifies the MobiLink worker thread that carried out the synchronization. The worker is identified as *n.m*, where *n* is the stream number and *m* is the thread number.
- ◆ **User** Identifies the synchronization user.
- ◆ **Version** The version of the synchronization script.
 ☞ For information about script versions, see [“Script versions” on page 49](#).
- ◆ **Start Time** The date and time when the MobiLink synchronization server started the synchronization. (This may be later than when the synchronization was requested by the client.)
- ◆ **Duration** The total duration of the synchronization, in seconds.
- ◆ **Verify** The time in seconds for MobiLink to validate the synchronization request, validate the user name, and validate the password (if your synchronization setup requires authentication).
- ◆ **Preload** The time in seconds for MobiLink to receive the uploaded data from the client.
- ◆ **Begin** The time in seconds to run your `begin_synchronization` script, if one was run.
- ◆ **Upload** The time in seconds to apply the upload to the consolidated database. This is the time between the `begin_upload` script and the `end_upload` script.
- ◆ **P.F.D.** The time in seconds to run your `prepare_for_download` script, if one was run.
- ◆ **Download** The time in seconds to download the data. This is the time between the `begin_download` script and the `end_download` script. If download acknowledgement is enabled, this includes the time to apply the download on the remote database and return acknowledgement.

- ◆ **End** The time in seconds to run the end_synchronization script, if one was run.

To sort the table by a specific column, click on the column heading.

You can close the Details Table pane by clearing View ► Details Table.

Chart pane

The Chart pane presents the same information as the Details Table, but in graphical format. The bars in the Chart represent the length of time taken by each synchronization, with subsections of the bars representing the phases of the synchronization.

Viewing data

Click a synchronization to select that synchronization in the Details Table.

Double-click a synchronization to open the Synchronization Session Properties for the synchronization. For more information, see [“Synchronization Properties” on page 306](#).

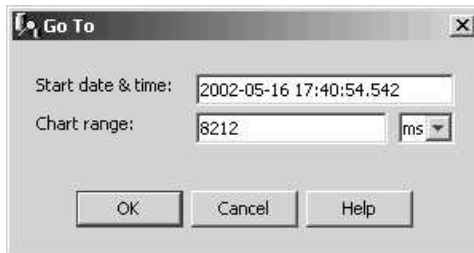
Grouping data by thread or user

You can group the data by worker thread or by user. Choose View ► By User or View ► By Worker Thread.

Zooming in on data

There are three ways to select the data that is visible:

- ◆ **Scrollbar** Click the scrollbar at the bottom of the Chart pane and slide it.
- ◆ **Go To dialog** Open this dialog by choosing View ► Go To. The Go To dialog appears:



Start Date & Time lets you specify the start time for the data that appears in the Chart pane. If you change this setting, you must specify at least the year, month, and date of the date-time.

Chart Range lets you specify the duration of time that is displayed. The chart range can be specified in milliseconds, seconds, minutes, hours, or days. The chart range determines the granularity of the data: a smaller length of time means that more detail is visible.

- ◆ **Overview Pane** The box in the Overview pane indicates the area being displayed in the Chart. It allows you to quickly select a portion of data to

view. You can easily resize or move the box to see different data, or see data at different granularity. If you make the box smaller you shorten the interval of the visible data in the Chart, which makes more detail visible. Click to move the current box without changing the zoom. Drag in the Overview to redraw the box and select a different zoom and position.

Time axis

At the bottom of the Chart pane there is a scale showing time periods. The format of the time is readjusted automatically depending on the span of time that is displayed. You can always see the complete date-time by hovering your cursor over the scale.

Default color scheme

You can view or set the colors in the Chart pane by opening the Options dialog (available from the Tools menu). The default color scheme for the Chart pane uses green for uploads, red for downloads, and blue for begin and end phases, with a darker shade for earlier parts of a phase.

☞ For information about setting colors, see [“Options” on page 305](#).

Overview pane

The Overview pane shows you an overview of the entire Monitor session. The area that is currently displayed in the Chart pane is represented as a box in the Overview. Click in the Overview pane to move the box (and thus move the start time of the data shown in the Chart) or drag in the Overview to redraw the box to change the box’s location and size (and thus change the start time and the range of data)

You can separate the Overview pane from the rest of the Monitor window. In the Options dialog, open the Overview tab and clear the Keep Overview Window Attached to Main Window checkbox.

☞ For more information, see [“Options” on page 305](#).

You can close the Overview pane by clearing View ► Overview Pane.

Options

Options allow you to specify a number of settings, including colors and patterns for the graphical display in the Chart pane (the middle pane of the MobiLink Monitor) and the Overview pane (the bottom pane).

To open the Options dialog, open the Monitor and choose Tools ► Options.

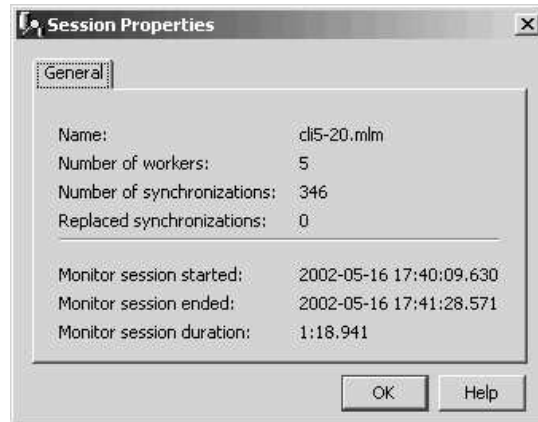
Restoring defaults

To restore default settings, delete the file `.mlMonitorSettings`. This file is stored in your user profiles directory.

Session Properties

The Session Properties dialog provides basic information about the monitoring session.

To open the Session Properties dialog, open the Monitor and choose File ► Properties. In the following example of a Session Properties dialog, data for a series of synchronizations has been saved in a file called *cli5-20.mlm*:



Synchronization Properties

Double-click a synchronization in either the Details Table or the Chart to see properties for that synchronization.

You can choose to see statistics for all tables (which is the sum for all tables in the synchronization), or for individual tables. The dropdown list provides a list of the tables that were involved in the synchronization.

☞ For an explanation of the statistics in Synchronization Properties, see [“MobiLink statistical properties” on page 310](#).

Saving Monitor data

You can save the data from a Monitor session as a binary file (.mlm) or as a text file with comma-separated values (.csv). To save the data, choose File ► Save As.

- ◆ Save the data as a binary (.mlm) file if you want to view the saved data in the MobiLink Monitor. To reopen, choose File ► Open.
- ◆ Save the data as a comma separated file (.csv) if you want to view it in another tool, such as Microsoft Excel. This will save all the information in the session and synchronization property sheets, except per table information and the session begin and end time. You can also open a .csv file in the Monitor.

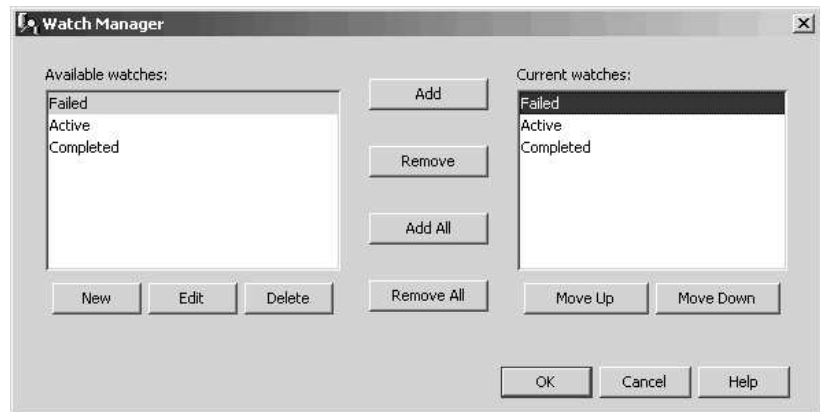
In the .csv file format, time durations are stored in milliseconds.

You can specify that you want data to be saved automatically to a file. To do this, choose Tools ► Options, and enter an output file name on the General tab. The output file is overwritten by new data.

Customizing your statistics

The Watch Manager allows you to visibly distinguish synchronizations that meet criteria that you specify. For example, you might want to highlight big synchronizations, long synchronizations, small synchronizations that take a long time, or synchronizations that receive warnings.

To open the Watch Manager, open the Monitor and then click Tools ► Watch Manager. The Watch Manager appears:



The left pane contains a list of all available watches. The right pane contains a list of active watches. To add or remove a watch from the active list, select a watch in the left pane and click the appropriate button.


There are three predefined watches (Active, Completed, and Failed). You can edit predefined watches to change the way they are displayed, and you can deactivate them by removing them from the right pane.

No synchronizations are displayed in the Chart unless they meet the conditions of a watch. If you disable all watches (by removing them from the Current Watches list), then no synchronizations are shown in the Chart or Overview.

The order of watches in the right pane is important. Watches that are closer to the top of the list are processed first. Use the Move Up and Move Down buttons to organize the order of watches in the right pane.

You can use the predefined watches, and create other watches. To edit a watch condition, remove it and then add the new watch condition.

❖ **To create a new watch**

1. In the Watch Manager, click New.
The New Watch dialog appears.
2. Give the watch a name in the Name box.
3. Select a property, comparison operator, and value.
 For a complete list of properties, see [“MobiLink statistical properties” on page 310](#).
4. Click Add. (You must click Add to save the settings.)
5. If desired, select another property, operator, and value, and click Add.
6. Select a pattern for the watch in the Chart pane. (The Chart pane is the middle pane in MobiLink Monitor.)
7. Select a color for the watch in the Overview pane. (The Overview pane is the bottom pane in the MobiLink Monitor.)

MobiLink statistical properties

Following is a list of the properties that are available in the MobiLink Monitor. These can be specified in the New Watch dialog. They can also be viewed in the Synchronization Properties dialog. In Synchronization Properties, the property names do not contain underscores.

Property	Notes
active	True if the synchronization is in progress.
begin_sync	Time for the begin_synchronization event.
completed	True if the synchronization completed successfully.
conflicted_deletes	Number of uploaded deletes for which conflicts were detected.
conflicted_inserts	Number of uploaded inserts for which conflicts were detected.
conflicted_updates	Number of uploaded updates for which conflicts were detected.
connection_retries	Number of times the MobiLink synchronization server retried the connection to the consolidated database.
download	Time for the download.
download_bytes	Bytes downloaded to the synchronization client.
download_deleted_rows	Number of row deletions fetched from the consolidated database by the MobiLink synchronization server (using download_delete_cursor scripts).
download_errors	Number of errors that occurred during the download.
download_fetched_rows	Number of rows fetched from the consolidated database by the MobiLink synchronization server (using download_cursor scripts).
download_filtered_rows	Number of fetched rows that were not downloaded to the MobiLink client because they matched rows that the client uploaded.

Property	Notes
download_warnings	Number of warnings that occurred during the download.
duration	Total time for the synchronization, as measured by the MobiLink synchronization server. This does not include time when the synchronization request is queued waiting for an available worker thread.
end_sync	Time for the end_synchronization event.
ignored_deletes	Number of uploaded deletes that were ignored.
ignored_inserts	Number of uploaded inserts that were ignored.
ignored_updates	Number of uploaded updates that were ignored.
preload_upload	Time for the transfer of the upload data from the client to the MobiLink synchronization server.
prepare_for_download	Time for the prepare_for_download event.
start_time	Date-time (in ISO-8601 extended format) for the start of the synchronization. All fields of the format must be specified: <i>YYYY-MM-DD hh:mm:ss.sss</i> or <i>YYYY-MM-DD hh:mm:ss.sss</i> , depending on your locale setting.
sync_deadlocks	Total number of deadlocks in the consolidated database that were detected for the synchronization.
sync_errors	Total number of errors that occurred for the synchronization.
sync_tables	Number of client tables that were involved in the synchronization.
sync_warnings	Total number of warnings that occurred for the synchronization.
upload	Time for data to be uploaded to the consolidated database.
upload_bytes	Number of bytes uploaded from the synchronization client.

Property	Notes
upload_deadlocks	Number of deadlocks in the consolidated database that were detected during the upload.
upload_deleted_rows	Number of row deletions that were uploaded from the synchronization client.
upload_errors	Number of errors that occurred during the upload.
upload_inserted_rows	Number of row insertions that were uploaded from the synchronization client.
upload_updated_rows	Number of row updates that were uploaded from the synchronization client.
upload_warnings	Number of warnings that occurred during the download.
user	Name of the MobiLink client.
verify_upload	Time for verifying the synchronization protocol and authenticating the synchronization client.
version	Name of the synchronization version.
worker	Identifier for the MobiLink worker thread used for the synchronization in the form $n.m$, where n is the stream number and m is the thread number.

CHAPTER 14

Synchronizing Through a Web Server

About this chapter

This chapter describes one way to route MobiLink synchronization through a web server. This method is particularly useful for synchronizing across a firewall or with multiple MobiLink synchronization servers.

The software that routes requests is called the Redirector.

Contents

Topic:	page
Introduction	314
Setting up the Redirector	315
Configuring MobiLink clients and servers for the Redirector	316
Configuring Redirector properties (all versions)	318
Configuring an NSAPI Redirector for Netscape web servers	320
Configuring an ISAPI Redirector for Microsoft web servers	323
Configuring the servlet Redirector	325

Introduction

MobiLink includes a web server extension called the **Redirector** that routes requests and responses between a client and the MobiLink synchronization server. A plug-in such as this is also commonly called a **reverse proxy**.

Using the Redirector, you can configure your web server to route specific URL requests to one or more computers running MobiLink synchronization server. The Redirector also implements load-balancing and failover: each MobiLink synchronization server is tested at set intervals and requests are no longer sent to a server that is not responding. It also detects when a MobiLink synchronization server is running again and resumes sending requests at that time.

Web servers can be configured to pass requests with specific URLs or ranges of URLs to extension programs commonly written in the form of perl CGI scripts, DLLs, or other extension mechanisms. These extension programs may access external data sources and provide responses for the web server to deliver to its clients.

Uses of the Redirector

This chapter describes one way to set up MobiLink synchronization across a firewall, with the MobiLink synchronization server running inside the firewall, and the MobiLink clients outside the firewall. Synchronization is routed through a web server.

The main reason for routing requests through a web server is to use existing web server and firewall configurations for HTTP or HTTPS synchronization. However, a web server can operate as a proxy without the Redirector. The Redirector is most useful when you have more than one MobiLink synchronization server.

HTTPS synchronization

In HTTPS synchronization, HTTP headers are encrypted over SSL/TLS using RSA encryption before being sent to or from the server. HTTPS is only used for the connection between the MobiLink client and the web server. The web server decrypts the HTTPS and sends HTTP to MobiLink via the Redirector.

The HTTPS stream is slower than other secure streams, so it is recommended that it be used only if the HTTPS protocol is required.

Supported web servers

Plug-ins are provided for the following web servers:

- ◆ Netscape iPlanet web servers (the NSAPI Redirector)
- ◆ Microsoft web servers (the ISAPI Redirector)
- ◆ Web servers that support the Java Servlet API 2.2 (the servlet Redirector)

Setting up the Redirector

The following sections describe how to configure your web server to manage synchronization requests.

❖ To set up synchronization through a web server

1. Configure the MobiLink clients and MobiLink synchronization server.
 - ☞ See [“Configuring MobiLink clients and servers for the Redirector” on page 316.](#)
2. Ensure that the Redirector configuration file is on the same computer as the web server.
 - ☞ See [“Configuring Redirector properties \(all versions\)” on page 318.](#)
3. Modify the Redirector configuration file.
 - ☞ See [“Configuring Redirector properties \(all versions\)” on page 318.](#)
4. Perform web server-specific configuration.
 - ☞ See:
 - ◆ [“Configuring an NSAPI Redirector for Netscape web servers” on page 320](#)
 - ◆ [“Configuring an ISAPI Redirector for Microsoft web servers” on page 323](#)
 - ◆ [“Configuring the servlet Redirector” on page 325](#)

Configuring MobiLink clients and servers for the Redirector

This section describes how to configure MobiLink clients and the MobiLink synchronization server for synchronization through a web server. The following procedure sets the parameters required for requests directed through web servers.

❖ To configure MobiLink clients and servers

1. Specify the communication type for the MobiLink clients. For example, HTTP or HTTPS protocol may be specified on the `dbmlsync` command line as follows, where `sync-type` is `http` or `https`.

dbmlsync -e ctp=sync-type

☞ For more information, see “CommunicationType (ctp) extended option” [*MobiLink Synchronization Reference*, page 46].

2. Set the following HTTP/HTTPS synchronization stream parameters on the MobiLink client:

- ◆ **host** the name or IP address of the web server.
- ◆ **port** the web server port accepting HTTP or HTTPS requests.
- ◆ **url_suffix** This setting depends on the type of web server you are using:

- For ISAPI web servers, set this to the following:

`exe_dir/iaredirect.dll/ml/`

where `exe_dir` is the location of `iaredirect.dll`.

- For NSAPI web servers, set this to the following:

`mlredirect/ml/`

where `mlredirect` is a name mapped in your `obj.conf` file.

- For servers that support the Java Servlet API 2.2, set this to the following:

`iaredirect/servlet/redirect/ml/`

☞ For UltraLite clients, for more information, see “HTTP stream parameters” [*UltraLite Database User's Guide*, page 184] and “HTTPS stream parameters” [*UltraLite Database User's Guide*, page 186].

☞ For Adaptive Server Anywhere clients, for more information, see “CREATE SYNCHRONIZATION USER statement [MobiLink]” [*ASA SQL Reference*, page 351].

3. The MobiLink server must be started with the HTTP protocol to use HTTP or HTTPS for communication between the client and the proxy. The Redirector cannot use HTTPS directly.

For example, the HTTP protocol may be specified on the dbmlsrv9 command line as follows:

dbmlsrv9 -x http

☞ For more information, see “-x option” [*MobiLink Synchronization Reference*, page 24].

4. Set the following parameters on the MobiLink server:
 - ◆ **port** for the HTTP protocol, MobiLink defaults to port 80. For the HTTPS protocol, MobiLink defaults to port 443. If the MobiLink synchronization server is running on the same machine as the web server, port 80 is normally in use by the web server. If this is the case you must specify a different port. For example, you could use port 2439, which is the Internet Assigned Numbers Authority (IANA)-registered port number for the MobiLink synchronization server.
 - ◆ **contd_timeout** This is the number of seconds to wait to receive the next part of a partially completed synchronization before the synchronization is abandoned. This setting is optional and has a default value of 30 seconds.

You may wish to increase the timeout parameters if your applications involve large synchronizations over slow networks.
5. Complete the steps in [“Configuring Redirector properties \(all versions\)” on page 318](#).

Configuring Redirector properties (all versions)

This section describes generic web server configuration steps to configure Redirector properties.

❖ To configure Redirector properties

1. Complete the steps in [“Configuring MobiLink clients and servers for the Redirector” on page 316](#).

2. Copy *redirector.config* to the web server.

The file *redirector.config* is provided with the MobiLink synchronization server installation, in the *MobiLink\redirector* subdirectory of your SQL Anywhere installation.

If the MobiLink synchronization server is not installed on the same computer as the web server, copy *redirector.config* to the computer that holds the web server.

For Microsoft web servers, copy *redirector.config* to the directory *Inetpub/scripts*. For other web servers, you can copy *redirector.config* to any directory.

3. Configure the Redirector configuration file.

To configure communications between the web server and MobiLink synchronization server, you must edit the file *redirector.config* on the computer that holds the web server.

You can set the following directives in this file:

- ◆ **LOG_LEVEL** used to control the amount of output written to the log file. Values are 0, 1, and 2, with 1 being the default and 2 generating the most output.
- ◆ **ML** used to list the computers running MobiLink synchronization server, in the form `ML=host:port`. **ML** is case sensitive.
- ◆ **ML_CLIENT_TIMEOUT** used to ensure that each step of a single synchronization is directed to the same MobiLink synchronization server. The default value is 600 seconds (ten minutes).

Information is maintained by the MobiLink synchronization server for the duration of a synchronization, so each step of a synchronization should be handled by the same server. The Redirector maintains an association between client and server for the duration of `ML_CLIENT_TIMEOUT`. The value of this parameter should be greater than the longest step in any user's synchronization.
- ◆ **REDIRECTOR_HOST** used to specify the machine name of the web server running the Redirector. For example, `myCompany.com`. If your

web server is running behind a proxy or load balancer, REDIRECTOR_HOST must specify the host name of the proxy or load balancer.

- ◆ **REDIRECTOR_PORT** used to specify the port of the web server running the Redirector. For example, 80. If your web server is running behind a proxy or load balancer, REDIRECTOR_PORT must specify the port number of the proxy or load balancer.
- ◆ **SLEEP** used to set the interval in seconds at which the Redirector checks that the servers are functioning. The default is 1800 (30 minutes). For example, SLEEP=3600. SLEEP is case sensitive. The following rules apply to *redirector.config*:
 - The maximum line length is 300 characters.
 - Comments start with the hash character (#).
 - You cannot include spaces or tabs in the directive definitions.

4. Complete the web server-specific configuration in one of the following sections:

- ◆ [“Configuring an ISAPI Redirector for Microsoft web servers” on page 323](#)
- ◆ [“Configuring an NSAPI Redirector for Netscape web servers” on page 320](#)
- ◆ [“Configuring the servlet Redirector” on page 325](#)

Example

Following is a sample *redirector.config* file. This file specifies the following:

- ◆ The Redirector should check every 1800 seconds that the servers are functioning.
- ◆ The three computers running MobiLink synchronization server that are able to process requests.
- ◆ The host name and port of the web server where the Redirector resides.

```
SLEEP=1800
ML=myServ-pc:80
ML=209.123.123.1:8080
ML=myCompany.com:8081
REDIRECTOR_HOST=test2.ianywhere.com
REDIRECTOR_PORT=8081
```

Configuring an NSAPI Redirector for Netscape web servers

The NSAPI Redirector is provided for the Netscape iPlanet Enterprise Edition web server. Following are setup instructions for the iPlanet web Server, Enterprise Edition 4.1, Service Pack 12.

❖ To configure NSAPI Redirector for iPlanet

1. Complete the steps in [“Configuring Redirector properties \(all versions\)” on page 318](#).
2. If necessary, copy the file *iaredirect.dll* to the computer that holds the web server. This file is installed with the MobiLink synchronization server, in the *MobiLink\redirector\nsapi* subdirectory of your SQL Anywhere installation.
3. Update the iPlanet web server configuration file *obj.conf* as follows.

Sample file provided

A complete sample copy of *obj.conf*, preconfigured for the MobiLink synchronization server, is provided in *MobiLink\redirector\nsapi*, and is called *obj.conf.example*. You can use this sample file to confirm where the following sections fit in to the file.

Note: For some versions of iPlanet, you may need to specify init directives in the *magnus.conf* file, rather than the *obj.conf* file. For an example, see the example at the end of this section.

Update the following sections of *obj.conf*.

- ◆ Specify where *iaredirect.dll* and *redirector.config* are located. At the end of the Init section, add the following text, where *<location>* is the actual location of the files. (*iaredirect.dll* and *redirector.config* can be in different locations, although both must be on the same computer as the web server.)

```
Init fn="load-modules" shlib="<location>/iaredirect.dll"
func="redirector_initialize_redirector"
Init fn="initialize_redirector"
    configFile="<location>/redirector.config"
```

- ◆ Specify the name of the Redirector to be used in URLs. At the beginning of the “default object” section, add the following text. This section should appear exactly as provided below, except that you can change *mlredirect* to whatever you wish. All requests of the form *http://host:port/mlredirect/ml/** will be sent to one of the MobiLink synchronization servers running with the Redirector.


```
<Object name=default>
NameTrans fn="assign-name" from="/mlredirect/ml/*"
          name="redirectToML"
```

- ◆ Specify the objects that are called by the Redirector. After the “default object” section, add two new objects, as follows:

```
<Object name="redirectToML">
Service fn="redirector" serverType="ml"
</Object>
```

4. Set the buffer size for the MobiLink upload streams.

Add a directive to your web server’s *magnus.conf* file to set the buffer size (in bytes) for the upload and download stream. For example:

```
ChunkedRequestBufferSize=2000000
```

This directive increases the buffer to 2 Mb. The value must be sufficient to accommodate the size of the uploaded data.

5. If you are using HTTPS synchronization, configure your server as follows:

- ◆ Start the iPlanet web server Administration Server.
Choose Start ► Programs ► iPlanet Web Server ► Start iWS Administration Server.
- ◆ Login to the Administration Server.
Choose Start ► Programs ► iPlanet Web Server ► Administer Web Server.
When prompted, enter your user ID and password.
- ◆ On the Servers tab, select your server from the list and click Manage.
- ◆ On the Security tab, click Request a Certificate.
- ◆ Generate a certificate request and have it signed by a certificate authority or using gencert, which requires a separate license.
 - To have the certificate request signed by a certificate authority, fill out the form.
 - To use the gencert utility, fill out the form, supplying your own e-mail address instead of the e-mail address of a certificate authority. Save the text of the certificate request to a file, then run the gencert utility. For more information, see “Certificate generation utility” [MobiLink Synchronization Reference, page 311].
- ◆ On the Security tab, click Install Certificate. Fill out the form and specify the location of your signed certificate.
- ◆ Click Manage Certificates to verify that your certificate has been installed correctly.

-
- ◆ On the Preferences tab, click Add Listen Socket. Specify the required parameters. The default port for HTTPS is 443. Select On from the Security dropdown list to activate HTTPS synchronization.

Example

Following are examples of the sections of *obj.conf* that configure the Netscape iPlanet web Server to route requests to MobiLink synchronization server.

```
Init fn="load-modules" shlib="D:/iaredirect.dll"
funcs="redirector,initialize_redirector"
Init fn=" initialize_redirector "
        configFile="D:/redirector.config"
# For iPlanet 6.0 service pack 1 the preceding Init lines should
# be
# placed in the magnus.conf file, rather than the obj.conf file.
...
<Object name=default>
NameTrans fn="assign-name" from="/mlredirect/ml/*"
        name="redirectToML"
...
<Object name="redirectToML">
Service fn="redirector" serverType="ml"
</Object>
```

❖ To test your configuration

1. Call the Redirector using the following syntax:

```
http://host:port/mlredirect/ml/
```

2. Check the log file to see if the Redirector logged a request.

Configuring an ISAPI Redirector for Microsoft web servers

If you are using a Microsoft web server, you can use the ISAPI version of the Redirector. Following are setup instructions for IIS 5.0.

❖ To configure ISAPI Redirector for Microsoft web servers

1. Complete the steps in [“Configuring Redirector properties \(all versions\)” on page 318](#).

2. Copy the file *iaredirect.dll* to *Inetpub/scripts* on the computer that holds the web server.

The file *iaredirect.dll* is installed with MobiLink synchronization server, in *MobiLink\redirector\isapi* under your SQL Anywhere directory

The directory *Inetpub/scripts* is in the Microsoft web server installation directory.

3. Copy the file *redirector.config* to *Inetpub/scripts* on the computer that holds the web server.
4. If you are using HTTPS synchronization, configure your server as follows:
 - ◆ Right-click My Computer and select Manage from the popup menu.
 - ◆ In the left pane, open the Services and Applications folder. Select Internet Information Services.
 - ◆ In the right pane, right-click the default web site and select Configure from the popup menu.
 - ◆ Click the Directory Security tab.
 - ◆ Click Server Certificate.
 - The Web Server Certificate wizard appears.
 - ◆ Select Create a New Certificate to generate a certificate request. Follow the remaining prompts, choosing to output the certificate request to a file.
 - ◆ Sign your certificate.
 - You can sign the certificate using a third-party certificate authority or using the gencert utility, which requires a separate license. For more information, see “Certificate generation utility” [*MobiLink Synchronization Reference*, page 311].
 - ◆ Click Server Certificate.
 - The Web Server Certificate wizard appears with different prompts to allow you to install the signed certificate. Follow the prompts.

-
- ◆ Click View Certificate to verify that your certificate has been correctly installed.

Note

The directory *Inetpub/scripts* is created during the web server installation with execute permissions. You can put *redirector.config* and *iaredirect.dll* in a different directory only if you use the IIS utility Internet Services Manager to give execute permissions to the directory.

❖ **To test your configuration**

1. Call the ISAPI Redirector using the following syntax:

protocol://host[:port]/exec_dir/iaredirect.dll/ml/

where:

- ◆ **protocol** is **http** or **https**.
- ◆ **host** is the host name of the web server.
- ◆ **port** is the port on which the web server is listening, if it is not the default port.
- ◆ **exec_dir** is the directory where you installed the Redirector dll, *iaredirect.dll*. The default directory is *scripts*.

For example,

`http://server:8080/scripts/iaredirect.dll/ml/`

2. Check the log file to see if the Redirector logged a request.

Configuring the servlet Redirector

The servlet version of the Redirector is supported for Apache Tomcat 4.0.6.

Configuring the servlet Redirector for Apache Tomcat servers

This section describes how to install the servlet version of the Redirector to work on an Apache web server in conjunction with the Tomcat servlet container. Testing of the Redirector software has been carried out using Tomcat version 4.1 and Apache 2.0.45.

Installation requires the following steps:

1. Complete the steps in “[Configuring Redirector properties \(all versions\)](#)” on page 318.
2. Install the servlet version of the Redirector in Tomcat.
3. Configure the Apache web server to run as a proxy.

This section uses `%CATALINA_HOME%` and `%APACHE_HOME%` as the root directory of your Tomcat and Apache installation respectively.

❖ To install the servlet Redirector in Tomcat

1. Install Tomcat as a standalone server.

You can download Tomcat binaries from the Jakarta project on the Apache web site at <http://jakarta.apache.org>.

2. Optionally, set the required Tomcat HTTP port.

Tomcat binds to port 8080 by default. If there is a conflict, perhaps because another web server is using this port,

- ◆ open the file: `%CATALINA_HOME%/conf/server.xml`
- ◆ search for 8080 (which is in a `<Connector>` tag).
- ◆ Change it to a port that is not in use.

3. Install the servlet Redirector as a web application.

- ◆ Copy `iaredirect.war` file to `%CATALINA_HOME%/webapps`
- ◆ Shutdown and restart Tomcat.
Tomcat expands the war file and creates the directory `iaredirect` for the Redirector web application.
- ◆ Edit the file
`%CATALINA_HOME%/webapps/iaredirect/WEB-INF/web.xml`.
Search for **redirector.config** (in an `<init-param>` tag), and correct the path for the `redirector.config` file.

Change the entry **redirector.config** to read *drive:/path/redirector.config*. Even on Windows operating systems, use a forward slash as a path separator, as in *d:/redirector.config*.

- ◆ Shutdown and restart Tomcat for the changes to take effect.
Once the changes have taken effect, you no longer need the war file in the deployed location.
- ◆ The Redirector can now be invoked through the following URL:
http://tc-machine:tc-port/iaredirect/servlet/redirect/ml/
where *tc-machine* is the machine and *tc-port* the port on which Tomcat is listening.

❖ To configure the Apache web server as a proxy

1. Install the Apache web server.

You can download binaries from the Apache web site at <http://www.apache.org>.

2. Optionally, change the Apache web server port.

Edit the file *%APACHE_HOME%/conf/httpd.conf* and change the **Port** setting to the desired port.

3. Configure Apache to run as a proxy.

In *%APACHE_HOME%/conf/httpd.conf*, add the following two directives:

```
LoadModule proxy_module {module-path}/mod_proxy.so
LoadModule proxy_connect_module {module-path}/mod_proxy_
    connect.so
LoadModule proxy_http_module {module-path}/mod_proxy_
    http.so
```

For example, the path may be *modules/mod_proxy.so* (the default).

4. Configure Apache to forward Redirector URLs to Tomcat.

In *%APACHE_HOME%/conf/httpd.conf*, add the following two directives so that Apache forwards URLs of the form *http://localhost/iaredirect/** to the Tomcat 4 Connector listening on port 8080:

```
ProxyPass /iaredirect http://localhost:8080/iaredirect
```

The port number must match the port number used for Tomcat. If Tomcat and Apache are not running on the same machine, provide the machine name where Tomcat is running instead of **localhost**.

5. If you are using HTTPS synchronization, configure your server as follows:
 - ◆ Download and install binaries for `mod_ssl` and `OpenSSL`. You can find them using the Apache Module Registry at <http://modules.apache.org/>. `mod_ssl.so` must be copied to `%APACHE_HOME%\modules`. `libeay32.dll` and `ssleay32.dll` must be copied to `%APACHE_HOME%\bin`.
 - ◆ Generate a server certificate and private key either by generating a request with `reqtool.exe` and sending it to a third party certificate authority to sign it, or by generating a certificate directly using `gencert.exe`. The private key can either be in the same file as the server certificate or in its own file.
 - ◆ Add the following lines to `%APACHE_HOME%\conf\httpd.conf`:

```
LoadModule ssl_module modules/mod_ssl.so
SSLEngine on
SSLCertificateFile certificate_file
```

where *certificate_file* is the path and file name of the server's certificate file.

If the server's private key is in a separate file from the server's certificate, add the additional line

```
SSLCertificateKeyFile private_key_file
```

where *private_key_file* is the path and file name of the server's private key.

If the private key is encrypted using a pass phrase and you are running under win32, add the additional line

```
SSLPassPhraseDialog exec:exe_name
```

where *exe_name* is the path and file name of an executable that will return the pass phrase on stdout.

Alternatively, the pass phrase can be removed from the private key using `openssl`:

```
openssl rsa -in src_file -out dst_file
```

where *src_file* is the path and file name of the private key protected by a pass phrase, and *dst_file* is the path and file name of the output file that will contain the unprotected private key. Note that this may reduce server security.

Example of HTTPS Configuration

Following is an example of how to configure Apache for HTTPS. This example uses Apache's virtual host feature to read HTTPS from port 443 (the default HTTPS port) and HTTP from port 80 at the same time.

```
LoadModule ssl_module modules/mod_ssl.so

Listen 80
Listen 443

NameVirtualHost *:443
<VirtualHost _default_:443>
    ServerName server_name:443
    ErrorLog logs/https_error
    CustomLog logs/https_access common

    SSLEngine on
    SSLCertificateFile rsaserver.crt
    SSLCertificateKeyFile rsaserver.key
</VirtualHost>
```

Verifying your setup

❖ To check your configuration

1. Call the Redirector using the following syntax:

```
http://host:port/iaredirect/servlet/redirect/ml/app
```

2. Check the log file to see if the Redirector logged a request.

CHAPTER 15

Running MobiLink Outside the Current Session

About this chapter

This chapter describes how to run the MobiLink synchronization server as a daemon or service.

You can set up MobiLink synchronization server to be available all the time. To make this easier, you can run the MobiLink synchronization server for Windows and for UNIX in such a way that, when you log off the computer it remains running. The way you do this depends on your operating system.

- ◆ **UNIX daemon** You can run the MobiLink synchronization server as a daemon using the `-ud` command line option, enabling the MobiLink server to run in the background, and to continue running after you log off.
- ◆ **Windows service** You can run the Windows MobiLink server as a service.

Contents

Topic:	page
Running the UNIX MobiLink server as a daemon	330
Running the Windows MobiLink server as a service	331
Troubleshooting MobiLink server startup	336

Running the UNIX MobiLink server as a daemon

To run the UNIX MobiLink server in the background, and to enable it to run independently of the current session, you run it as a **daemon**.

❖ To run the UNIX MobiLink server as a daemon

1. Use the `-ud` command line option when starting the MobiLink server.
For example:

```
dbmlsrv9 -c "dsn=ASA 9.0 Sample;uid=DBA;pwd=SQL" -ud
```

☞ For more information, see “`-ud` option” [*MobiLink Synchronization Reference*, page 20].

Running the Windows MobiLink server as a service

To run the Windows MobiLink server in the background, and to enable it to run independently of the current session, you run it as a **service**.

You can carry out the following service management tasks from the command line, or on the Services tab in Sybase Central:

- ◆ Add, edit, and remove services.
- ◆ Start, stop, and pause services.
- ◆ Modify the parameters governing a service.
- ◆ Add databases to a service, so you can run several databases at one time.

Adding, modifying, and removing services

The service icons in Sybase Central display the current state of each service using a traffic light icon that displays running, paused, or stopped.

❖ To add a new service (Sybase Central)

1. In Sybase Central, click the server in the left pane, and then open the Services tab in the right pane.
2. Double-click Add Service.
3. Follow the instructions in the wizard.

You can also use the `dbsvc` utility to create the service. For more information, see “Managing services using the `dbsvc` command-line utility” [ASA Database Administration Guide, page 519].

❖ To remove a service (Sybase Central)

1. In Sybase Central, click the server in the left pane, and then open the Services tab in the right pane.
2. In the right pane, right-click the icon of the service you want to remove and choose Delete from the popup menu.

❖ To change the parameters for a service

1. In Sybase Central, click the server in the left pane, and then open the Services tab in the right pane.
2. In the right pane, right-click the service you want to change and choose Properties from the popup menu.
3. Alter the parameters as needed on the tabs of the Service property sheet.
4. Click OK when finished.

Changes to a service configuration take effect the next time the service is started.

Setting the startup option The following options govern startup behavior for MobiLink services. You can set them on the General tab of the service property sheet.

- ◆ **Automatic** If you choose **Automatic**, the service starts whenever the Windows operating system starts. This setting is appropriate for database servers and other applications running all the time.
- ◆ **Manual** If you choose **Manual**, the service starts only when a user with Administrator permissions starts it. For information about Administrator permissions, see your Windows documentation.
- ◆ **Disabled** If you choose **Disabled**, the service will not start.

The startup option is applied the next time Windows is started.

Specifying command line options The Configuration tab of the service property sheet provides a text box for typing command line options for a service. Do not type the name of the program executable in this box.

For example, to start a MobiLink synchronization service with verbose logging and three worker threads, type the following in the Parameters box:

```
-c "dsn=ASA 9.0 Sample;uid=DBA;pwd=SQL"  
-vc  
-w 3
```

☞ The command line options for a service are the same as those for the executable. For a full description of the command line options for MobiLink, see “MobiLink synchronization server” [*MobiLink Synchronization Reference*, page 4].

Setting account options You can choose which account the service runs under. Most services run under the special LocalSystem account, which is the default option for services. You can set the service to log on under another account by opening

the Account tab on the Service property sheet, and typing the account information.

If you choose to run the service under an account other than LocalSystem, that account must have the “log on as a service” privilege. This can be granted from the Windows User Manager application, under Advanced Privileges.

Whether or not an icon for the service appears on the taskbar or desktop depends on the account you select, and whether Allow Service to Interact with Desktop is checked, as follows:

- ◆ If a service runs under LocalSystem, and Allow Service to Interact with Desktop is checked in the service property sheet, an icon appears on the desktop of every user logged in to Windows NT/2000/XP on the computer running the service. Consequently, any user can open the application window and stop the program running as a service.
- ◆ If a service runs under LocalSystem, and Allow Service to Interact with Desktop is unchecked in the service property sheet, no icon appears on the desktop for any user. Only users with permissions to change the state of services can stop the service.
- ◆ If a service runs under another account, no icon appears on the desktop. Only users with permissions to change the state of services can stop the service.

Changing the executable file

To change the program executable file associated with a service in Sybase Central, click the Configuration tab on the Service property sheet and type the new path and file name in the File Name box.

If you move an executable file to a new directory, you must modify this entry.

Starting, stopping, and pausing services

❖ **To start, stop, or pause a service**

1. In Sybase Central, click the server in the left pane, and then open the Services tab in the right pane.
2. Right-click the service and choose Start, Stop, or Pause from the popup menu.

To resume a paused service, right-click the service and select Continue from the popup menu.

If you start a service, it keeps running until you stop it. Closing Sybase Central or logging off does not stop the service.

Stopping a service closes all connections to the database and stops the database server. For other applications, the program closes down.

Pausing a service prevents any further action being taken by the application. It does not shut the application down or (in the case of server services) close any client connections to the database. Most users do not need to pause their services.

Running more than one service at a time

Although you can use the Windows Service Manager in the Control Panel for some tasks, you cannot install or configure a MobiLink service from the Windows Service Manager. You can use Sybase Central to carry out all the service management for MobiLink.

When you open the Windows Service Manager from the Windows Control Panel, a list of services appears. The names of the Adaptive Server Anywhere services are formed from the Service Name you provided when installing the service, prefixed by Adaptive Server Anywhere. All the installed services appear together in the list.

This section describes topics specific to running more than one service at a time.

Service dependencies

In some circumstances you may wish to run more than one executable as a service, and these executables may depend on each other. For example, you must run the MobiLink synchronization server and the database server in order to synchronize.

In cases such as these, the services must start in the proper order. If a MobiLink synchronization service starts up before the consolidated database server has started, it fails because it cannot find the consolidated database server. The sequence must be such that the database server is running when you start the MobiLink server. (This does not apply if the consolidated database server is on another computer.)

You can prevent these problems using service groups, which you manage from Sybase Central.

Service groups

You can assign each service on your system to be a member of a service group. By default, each service belongs to a group. The default group for the MobiLink synchronization server is ASANYMobiLink.

Before you can configure your services to ensure they start in the correct order, you must check that your service is a member of an appropriate group. You can check which group a service belongs to, and change this group, from Sybase Central.

❖ **To check and change which group a service belongs to**

1. In Sybase Central, click the server in the left pane, and then open the Services tab in the right pane.
2. Right-click the service and choose Properties from the popup menu.
3. Click the Dependencies tab. The top text box displays the name of the group the service belongs to.
4. Click Change to display a list of available groups on your system.
5. Select one of the groups, or type a name for a new group.
6. Click OK to assign the service to that group.

Managing service dependencies

With Sybase Central, you can specify dependencies for a service. For example:

- ◆ You can ensure that at least one group has started before the current service.
- ◆ You can ensure that any service starts before the current service.

❖ **To add a service or group to a list of dependencies**

1. In Sybase Central, click the server in the left pane, and then open the Services tab in the right pane
2. Right-click the service and choose Properties from the popup menu.
3. Click the Dependencies tab.
4. Click Add Services or Add Service Groups to add a service or group to the list of dependencies.
5. Select one of the services or groups from the list.
6. Click OK to add the service or group to the list of dependencies.

Troubleshooting MobiLink server startup

This section describes some common problems when starting the MobiLink server.

Ensure that network communication software is running

Appropriate network communication software must be installed and running before you run the MobiLink server. If you are running reliable network software with just one network installed, this should be straightforward. You should confirm that other software requiring network communications is working properly before running the MobiLink server.

If you are running under the TCP/IP protocol, you may want to confirm that ping and telnet are working properly. The ping and telnet applications are provided with many TCP/IP protocol stacks.

Debugging network communications startup problems

If you are having problems establishing a connection across a network, you can use debugging options at both client and server to diagnose problems. The startup information appears on the server window: you can use the `-o` option to log the results to an output file.


CHAPTER 16

Transport-Layer Security

About this chapter

This chapter describes transport-layer security (TLS). This security mechanism protects messages as they travel between a MobiLink client and the MobiLink synchronization server or between a database client and the database server.

Transport-layer security is a separately licensable component and must be ordered before you can install it. To order this component, see the card in your SQL Anywhere Studio package or see <http://www.sybase.com/detail?id=1015780>.

 For information about Adaptive Server Anywhere database security, see “Keeping Your Data Secure” [*SQL Anywhere Studio Security Guide*, page 3].

Contents

Topic:	page
About transport-layer security	338
Invoking transport-layer security	346
Certificate authorities	351
Certificate chains	352
Enterprise root certificates	353
Globally signed certificates	358
Obtaining server-authentication certificates	360
Verifying certificate fields	363

About transport-layer security

MobiLink transport-layer security uses encryption to protect the confidentiality and integrity of the synchronization data stream as it passes between a MobiLink client and the MobiLink synchronization server. This feature is important whenever this communication must travel over a public or wireless network. Under such circumstances, someone with a suitable radio or network connection could otherwise intercept your data.

Furthermore, transport-layer security allows a client application to verify the identity of a MobiLink synchronization server. Hence, client applications can ensure that they synchronize only with MobiLink synchronization servers they trust.

This security is implemented by means of digital certificates. You can achieve a variety of security objectives using different types of certificates and configuring them in different ways. This section introduces the concepts that underlie public-key cryptography and explains how they apply to digital certificates. Examples illustrate several typical arrangements, each offering different benefits.

MobiLink transport-layer security is implemented using Certicom encryption technology. This public-key cryptographic technology uses an RSA cipher suite or an elliptic-curve cipher suite. When transport-layer security is invoked, all messages sent between the client and server are encrypted using a 128-bit cipher.

Invoking transport-layer security

To invoke the server authentication features, you create and use digital certificates. Different types of certificates and different arrangements of these certificates allow you to provide various levels of security. You create the certificates using tools included with SQL Anywhere Studio.

About public-key cryptography

Public key cryptography makes use of mathematical systems that work with pairs of very large, associated numbers. These numbers, called **keys**, have particular properties. Each key can be used to encrypt information. Once encrypted, these messages can only be decrypted using the matching key.

One of the keys, called the **public key**, is published in a public forum. It can be used to encrypt information to be sent to the owner of the public key. The owner keeps the second key, called the **private key**, secret. A message encrypted with the public key can be decrypted only using the matching private key. Since the public key is published, anyone can create a message that only the owner of the private key can read.

In addition, a message encrypted with the private key can be decrypted by anyone who knows the public key. Such a message can be created only by someone who knows the private key. If the private key is kept secret, the owner can prove his or her identity by constructing such a message.

It is essential that the private key cannot be found easily through knowledge of the public key. The ease with which the private key can be derived from the public key is often associated with the strength of the cryptosystem and the size (in bits) of the public key. Another aspect of the private key is that it must be difficult to guess. The generation of high-quality private keys must incorporate pseudo-random data of high quality. If the data is predictable, it is easier for an adversary to guess the keys. To meet this criterion, the tools provided with MobiLink gather pseudo-random data from the operating system when generating new private and public key pairs.

The role of public-key cryptography

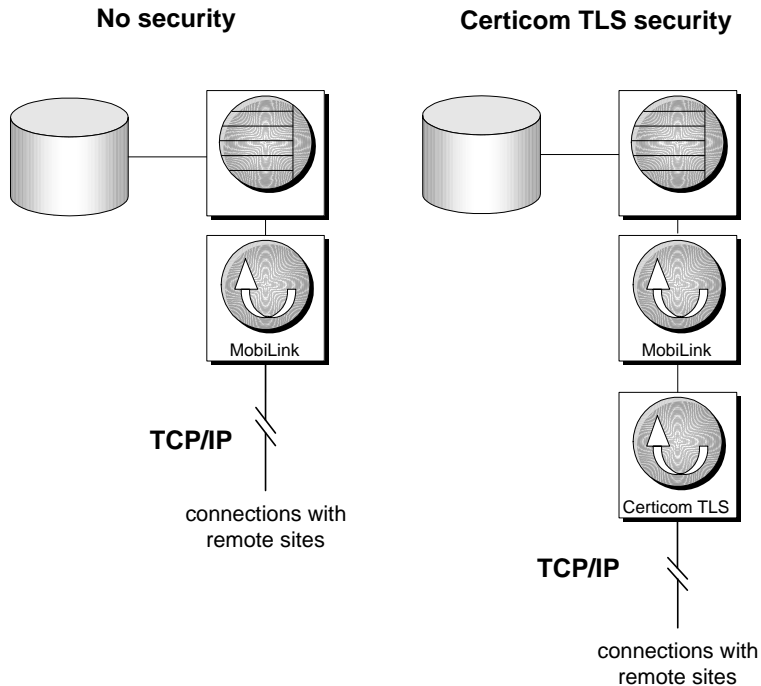
Public-key cryptography has many advantages. Using the public key, anyone can send a message that can be read only by the person who knows the matching private key. Likewise, someone can prove that they know a private key by using it to encrypt a message. To verify the identity of a key owner, you can send an arbitrary message and ask them to encrypt. You can be sure that person knows the private key if you can decrypt the resulting message with their public key.

These features make public-key cryptography especially useful when establishing a secure communication link and happen automatically when you establish a synchronization connection using transport-layer security.

Once the secure link is established, the server and client automatically switch to a symmetric-key system of equivalent strength. In a symmetric system, the same key is used to encrypt and decrypt messages. This type of symmetric cipher can be computed more efficiently, reducing the computation time required to encrypt and decrypt messages.

How transport-layer security works

Transport-layer security works by filtering all incoming and all out-going communication through the cipher of your choice. The translation occurs between the MobiLink synchronization server and the communication protocol of your choice. For example, adding security to a TCP/IP connection affects the architecture as shown in the following diagram:



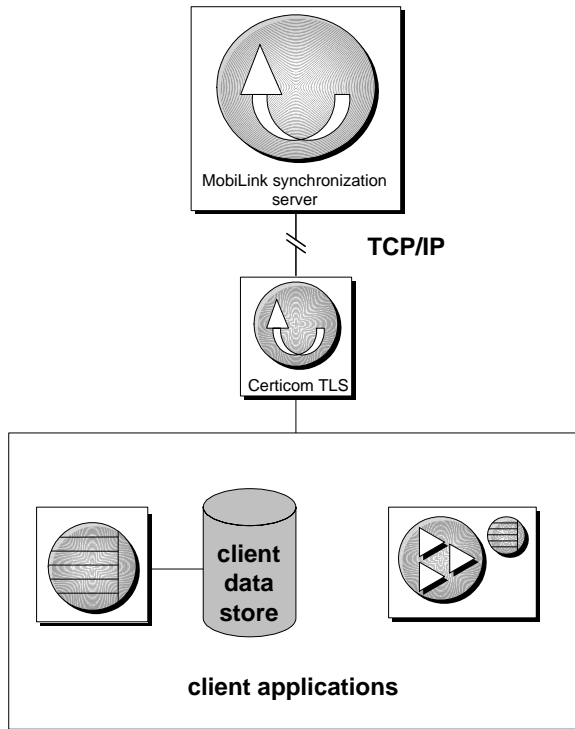
Transport-level security requires additional communication between a MobiLink client and the MobiLink synchronization server *before* the upload stream is sent. When a client initiates synchronization, it passes a message to the server. The client encrypts this message using the server's public key. The server decrypts this message using its private key. Initially, the server encrypts all messages to the client using the client's public key.

While this public-key/private-key cipher is secure as long as the private keys are kept secret, the encryption and decryption process is computationally intensive. To make further communication more efficient, the client and server agree upon and exchange another key and switch to a symmetric key cipher. They use this key and cipher for the rest of their communication because the symmetric cipher allows data to be encrypted and decrypted more efficiently.

Client architecture

To synchronize with a MobiLink synchronization server by secure means, the client must use the same cipher suite as the server. All messages received from the server via a communication protocol, such as TCP/IP, are decrypted before being passed to the remote MobiLink client. The following diagram

depicts how Certicom TLS cipher suite is added to a client using TCP/IP to communicate with a MobiLink synchronization server:



Digital certificates

A **digital certificate** is an electronic document that identifies a person or entity and contains a copy of their public key. Each certificate includes a public key so that anyone can communicate securely with the person or entity by encrypting information with this public key. Digital certificates conform to a standardized file format that contains the following information:

- ◆ Identity information, such as the name and address of the certificate owner.
- ◆ Public key.
- ◆ Expiry date.
- ◆ One or more digital signatures.

Digital signatures

A **digital signature** provides a means to detect whether a certificate has

been altered. A digital signature is a cryptographic operation created by calculating a value, called a **message digest**, from the identity information and the public key.

A message digest is a bit-value designed to change if any part of the certificate changes. The algorithm used to calculate the message digest is known to all users of the certificates. The correct value is encrypted with the private key contained in the certificate. Thus, anyone can detect alteration using the algorithm to calculate the message digest, using the public key to decrypt the message digest contained in the certificate, and comparing the two values.

A certificate constructed in this manner is called a **self-signed certificate** because the digital signature is constructed with the matching private key. Such a certificate cannot be altered without knowledge of the private key.

The importance of digital certificates

Digital certificates play the role of identity cards. The signatures prevent alteration because as long as the private keys used to create the signatures are kept secret, the digital certificate cannot be altered.

The role of digital certificates

A MobiLink synchronization server must be able to identify itself to clients with its own server certificate. The client must ensure that the certificate is authentic. To do so, the client must already have a trusted copy of the public certificate. Alternatively, the server's certificate may be signed by another certificate. In the latter case, the client must have a reliable copy of the signing certificate.

The MobiLink synchronization server must have access to its public certificate and to the private key for this certificate. This information is contained in a **server identity**. A server certificate is constructed by appending the private key to the matching public certificate.

The following figure displays a sample server certificate. This certificate is a server identity, suitable for use by a MobiLink synchronization server. This particular certificate has been signed by another certificate. The file contains both public certificates and the server's password.

```

-----BEGIN CERTIFICATE-----
MIIBqDCCAWSgAwIBAgIFMTIzNDUwCwYHKoZIzj0EAQUAMGsxDDAKBgNVBAYTA1VT
QTELMakGALUECBMCQ0ExEzARBgNVBAcTCkVtZXJ5dm1sbGUxPDASBgNVBAoUC1N5
YmFzZSBjbmMuMQ8wDQYDVQQLFAZTeWJhc2UxZj0EQBgNVBAMUCVN5YmFzZSBQDQAE
Fw05OTEwMTcxODAlMzZaFw0wOTEwMTcxODAlMzZaMGcxDDAKBgNVBAYTA1VTQTELM
akGALUECBMCQ0ExEzARBgNVBAcTCkVtZXJ5dm1sbGUxPDASBgNVBAoUC1N5YmFz
ZSBjbmMuMQ8wDQYDVQQLFANNRUMxETAPBgNVBAMUCE1vYm1saW5rMCswEAYHKoZI
zj0CAQYFK4EEAAEDFwACAx0L37T06bGehBNIRVJcma/Y0h5xoyYwJDAOBgNVHQ8B
Af8EBAMCAf4wEgYDVR0TAQH/BAgwBgEB/wIBCAjALBgqhkhjOPQBBQADMQAwwLgIV
Ad+41luT7/1URk7SfZTTiYgnR/rAAhUCCQRGc62100Mtt69TxuswBvI2OY=
-----END CERTIFICATE-----

-----BEGIN CERTIFICATE-----
MIIBrDCCAWIgaWIBAgIFMTIzNDUwCwYHKoZIzj0EAQUAMGsxDDAKBgNVBAYTA1VT
QTELMakGALUECBMCQ0ExEzARBgNVBAcTCkVtZXJ5dm1sbGUxPDASBgNVBAoUC1N5
YmFzZSBjbmMuMQ8wDQYDVQQLFAZTeWJhc2UxZj0EQBgNVBAMUCVN5YmFzZSBQDQAE
Fw05OTEwMTcxODAlMzZaFw0wOTEwMTcxODAlMzZaMGcxDDAKBgNVBAYTA1VTQTELM
akGALUECBMCQ0ExEzARBgNVBAcTCkVtZXJ5dm1sbGUxPDASBgNVBAoUC1N5YmFz
ZSBjbmMuMQ8wDQYDVQQLFAZTeWJhc2UxZj0EQBgNVBAMUCVN5YmFzZSBQDQAE
ByqGSM49AgEGBSuBBAAcAAAgFUVb7gQh0cy6XgxsRQUPaMCMiYk6MmMCQwDgYD
VR0PAQH/BAQDAgH+MBIGALUdEwEB/wQIMAYBAf8CAQowCwYHKoZIzj0EAQUAAzEA
MC4CFQITRvY7k6c3jy37KyC4iDj6UNGWnQIvA/gAja8SA2W7SyAfQ23oCY7n29Ss
-----END CERTIFICATE-----

-----BEGIN ENCRYPTED PRIVATE KEY-----
ME4wGgYJKoZIhvcNAQUDMA0ECL+NqY7WeMr/AgEFBDAZTKkSudCw2sUC45GKqATr
xc1epiZwr9g5jm6wK8cCqQBfgZxs/Ne8eC2an2klq1M=
-----END ENCRYPTED PRIVATE KEY-----

```

root certificate

server's certificate

server's private key
(encrypted with password)

Since other users may have access to the computer running the MobiLink synchronization server, the file containing the private key is protected by a password. This password is intended to maintain the honesty of the people given access to the computer. It does not provide an adequate barrier to an outside attack as the password is only a few characters in length. To further protect the private key, outsiders must be denied access to the MobiLink synchronization server by a firewall, or by other traditional means.

Instead of being signed directly by the certificate authority, the server's certificate may be the first certificate in a certificate chain. In this case, the client must trust the owners of all certificates and must have a trusted copy of the final certificate in the chain, called the root certificate. Such a certificate file would have a structure similar to that displayed above, but could contain a longer list of certificates.

Using chains of certificates

A certificate may be signed by other certificates, or it may be **self-signed**, which means it is signed only with its own private key. A sequence of public certificates, each signed by the next, is called a **certificate chain**. At one end of a typical chain is a certificate used for a particular MobiLink synchronization server. At the other end is a certificate, signed by no other certificates, called the **root certificate**.

You can arrange certificates in various ways, depending on your

requirements. The following sections describe how to construct and use certificate chains to achieve particular security goals. The following topics are covered:

- ◆ If you have only a single server, the simplest setup is to create a self-signed certificate. The only disadvantage is that the private key for the certificate must be held on the synchronization server, where it is harder to protect.
- ◆ An enterprise root certificate is of particular benefit to organizations using more than one MobiLink synchronization server. In this setup, MobiLink clients need keep only a copy of this root certificate to recognize any MobiLink synchronization server issuing a certificate signed by this root certificate.
- ◆ Commercial certificate authorities can benefit organizations that require the utmost in security. These organizations can help in two ways. First, the root certificates they use are of the highest possible quality, making these certificates somewhat less prone to attack. Secondly, commercial certificate authorities can provide a trusted third party when two companies wish to communicate securely but are not familiar with each other.
- ◆ You can, and in some cases should, use the facilities provided to verify certificate fields. This precaution is appropriate in many scenarios, but is particularly so when using a globally signed certificate. In this case, you are unlikely to want your clients to trust certificates that your certificate authority has signed for other customers.

In all cases, you must ensure that the MobiLink command line and log file are secure. This is best done using a firewall and by otherwise limiting access to the computer running the MobiLink synchronization server.

MobiLink transport-layer security is a flexible mechanism that lets you achieve the security important to your setup. The basic system allows you to keep information private, while certificates ensure MobiLink clients that they are talking to a trusted MobiLink synchronization server.

Server authentication

One method of breaking a system is to masquerade as the server. The client connects to what it thinks is the server, but the connection is unknowingly made to another, hostile server. To guard against this form of attack, the server can use a digital certificate. A digital certificate plays the role of an identity card.

Each digital certificate contains a public encryption key and information about the owner's identity. The certificates are designed in such a way that they can be altered only by someone who knows the matching private key. As long as this private key is kept a secret, clients can safely assume the identity information accurately identifies a server. To ensure that they are talking to the correct server, clients ask the server to prove that it knows the matching private key. The server can do so by decrypting a message that has been encrypted with the public key shown in the certificate.

Invoking transport-layer security

You can use transport-layer security when using the TCP/IP, HTTP, or HTTPS communication protocols. For TCP/IP and HTTP, you can use either RSA or elliptic-curve encryption. For HTTPS, you must use RSA encryption.

To invoke transport-layer security, you must first set it up for the client, storing the settings in the publication, subscription, or MobiLink user. You invoke server authentication on the dbmlsrv9 command line.

☞ For information about how to invoke transport-layer security on Adaptive Server Anywhere clients, see “CREATE SYNCHRONIZATION USER statement [MobiLink]” [*ASA SQL Reference*, page 351].

☞ For information about how to invoke server authentication for UltraLite clients, see “Synchronization for UltraLite Applications” [*UltraLite Database User's Guide*, page 143].

☞ For information about how to invoke server authentication for Adaptive Server Anywhere, see “-x option” [*MobiLink Synchronization Reference*, page 24].

The Certicom security software built into MobiLink uses certificates for the purpose of server identification. Two sample certificates are provided with Adaptive Server Anywhere, for elliptic-curve and for RSA encryption. The sample elliptic-curve certificate is called sample.crt and the password is tJ1#m6+W. The sample RSA certificate is called rsaserver.crt and the password is test.

Caution

The sample certificates should be used for testing purposes only. The sample certificates provide no security in deployed situations because they and their corresponding passwords are widely distributed with Sybase software. To protect your system, you must create your own certificate.

Confirming proper
startup

The MobiLink synchronization server screen displays informational messages on startup. These messages are also sent to the log file if you start the server with the -o option. You can use the -v+ option to provide more detailed messages.

If Certicom security starts properly, the informational messages confirm this fact. The absence of such messages indicates that Certicom security has not started properly.

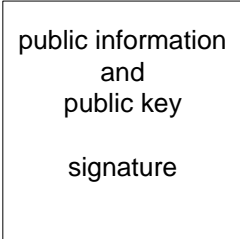
Self-signed certificates

SQL Anywhere Studio includes tools for working with certificates. These are included in the distribution if your license permits it. If so, you can choose to install these security components.

A utility named `gencert` allows you to generate new certificates. Since certificates are normally written in a machine-readable format, another utility, named `readcert`, displays the contents of a certificate in human-readable format.

You can make a number of types of certificates with the `gencert` utility. The easiest type to make is a self-signed (root) certificate, as no other signing certificate is required.

Self-signed public certificate



A rectangular box containing the following text:

public information
and
public key

signature

Use matching server
identity with one MobiLink
synchronization server

Give a trusted copy of the
public certificate to each
client

The main advantage of a setup with only one root certificate is simplicity; you need create only one certificate. This setup is often sufficient for simple setups involving only one MobiLink synchronization server. If you operate multiple MobiLink synchronization servers, an enterprise level certificate, discussed later, is often more convenient.

The biggest disadvantage is that a self-signed certificate is easier than other types to forge. This type of attack can be accomplished by creating a counterfeit certificate using a different key pair. Other types of certificates are more secure because they bear more than one digital signature.

Making a new self-signed certificate

To generate a root certificate, start the gencert utility from a command prompt using the `-r` option. The utility prompts you to enter the identity information, the certificate password and expiry date, and the names of the new certificate files.

In the following procedure, you are prompted for names for the certificate, private key, and server identity files. MobiLink accepts any name and extension for these files. However, Windows only recognizes `.crt` and `.cer` extensions as certificate files.

In the following procedure, an RSA certificate is generated. Alternatively, you can generate an elliptic-curve certificate by choosing certificate type ECC.

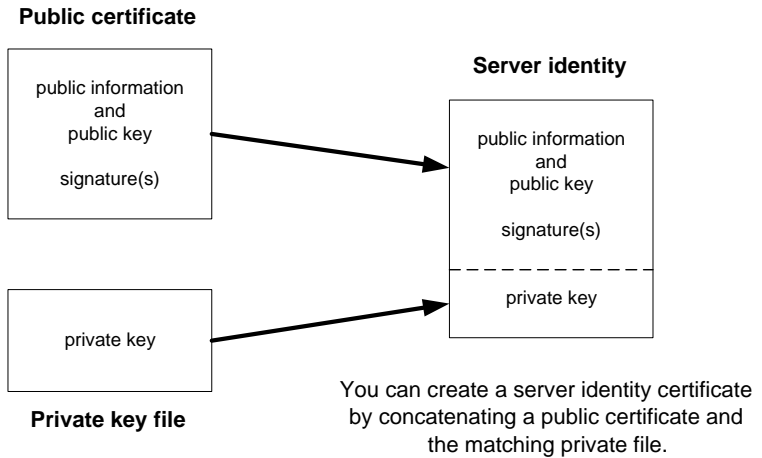
```
>gencert -r
Certificate Generation Tool
Choose certificate type ((R)SA or (E)CC): R
Enter key length (512-2048): 2048
Generating key pair...
Country: CA
State/Province: Ontario
Locality: Waterloo
Organization: Sybase, Inc.
Organizational Unit: IAS
Common Name: MobiLink
Serial Number: 2003.07.29.01
Certificate valid for how many years: 2
Enter password to protect private key: password
Enter file path to save certificate: self.crt
Enter file path to save private key: self.pri
Enter file path to save server identity: serv1.crt
```

The response to each question should be a string, except for the number of years to the expiry date, which must be an integer.

The utility creates three files, which in this example are called *self.crt*, *self.pri*, and *serv1.crt*.

- ◆ **self.crt** This file contains the new certificate, including the identity information, public key, expiry date, and signature. You can give out copies of this file to people whom you wish to contact you.
- ◆ **self.pri** This file contains the private key that matches the public key encoded in the certificate. The private key is encoded using the password you supplied, providing a modest barrier to others with access to your computer. However, since password encryption is not very secure, you must restrict access to this file to maintain secrecy.

- ◆ **serv1.crt** This file contains the same information as the above two files, combined into one file. It is intended for use with a MobiLink synchronization server. The server sends the public information to identify itself to clients. It requires the private key to decode messages returned by the clients. You must restrict access to this file. It, too, contains a copy of the private key, protected only by the password.



The server certificate contains the information in the public and private certificate files. You can make a server certificate by concatenating a public certificate and the file containing the private key.

Using a self-signed certificate

You can use the self-signed certificate for server authentication by following these steps:

1. Supply a copy of the public certificate to all clients. When the client first contacts the MobiLink synchronization server, the server will send them a copy of the public certificate, *self.crt*. The client can detect fake certificates by comparing the one sent by the server with the copy the client already has.
2. Tell each client that it is to trust only servers that can decrypt messages encoded using the public key contained within the copy of the supplied public certificate. For Adaptive Server Anywhere clients, you do so using the **trusted_certificates** security parameter. For example, you can tell an Adaptive Server Anywhere client to trust only the *self.crt* certificate by including the following parameter in the address clause of the synchronization subscription:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
FOR 'user001'
TO test -pub
ADDRESS 'host=myhost;security=ecc_tls (
    trusted_certificates=self.crt )'
```

To tell an UltraLite client to trust only the desired certificate, name the trusted certificate using the `-r` option when running the UltraLite generator, as follows. Open a command prompt and run the following command line:

```
ulgen -c "dsn=UltraLite 9.0 Sample;uid=DBA;pwd=SQL"
-r self.crt -j custapi
```

3. When you start the MobiLink synchronization server, specify the name of the server certificate file, `serv1.crt`, and the corresponding password. Open a command prompt and run the following command line:

```
dbmlsrv9 -c "dsn=UltraLite 9.0 Sample;uid=DBA;pwd=SQL"
-x tcpip ( security=ecc_tls ( certificate=serv1.crt;
    certificate_password=password ) )
```

You can hide the contents of the command line using the File Hiding utility, `dbfhide`. For more information, see “The File Hiding utility” [*ASA Database Administration Guide*, page 466].

Note that the clients do not need and should not have either the private key or the password that unlocks it. Clients need only the public certificate.

In contrast, the MobiLink synchronization server requires access to the private key, as well as to the public parts of the certificate. Thus, the server requires access to the server certificate file, which contains both public and private information.

The MobiLink synchronization server must have access to the private key and the password that protects it. For this reason, you must ensure that the MobiLink command line and log file are secure. This is best done using a firewall and by otherwise limiting access to the computer running the MobiLink synchronization server.

Note:

The certificate file name and password are not displayed in the log file.

Certificate authorities

One problem with self-signed certificates is that an adversary can create a fake certificate using a different public- and private-key pair. Someone, mistaking the fake certificate for the original, may unknowingly encrypt his or her message using the substitute public key, rather than that owned by the intended recipient. Only the adversary, who knows the substitute private key, could read a message encrypted using the fake certificate.

To guard against such an attack, both the user and the owner of the certificate must agree to trust a third party. This third party, called a **signing authority** or **certificate authority**, adds a digital signature to the certificate using his or her private key. Once signed, the document certificate can be altered only with the aid of the third party. To sign a certificate, the certificate authority need not know the private key of the certificate owner.

The certificate authority need not be an external person or organization. If the certificates are to be used only within the company, it may be appropriate for someone at the company to act as the certificate authority.

To create a trustworthy system, a certificate authority must confirm the identity of a certificate owner before signing a certificate. In particular, the certificate authority must check that the identity fields in the certificate accurately describe the certificate owner and that the certificate owner owns the matching private key.

Someone wishing to use this certificate to communicate with the certificate owner must have confidence in the following:

- ◆ Before signing the certificate, the certificate authority made certain that the identity information contained in the certificate correctly identified the certificate owner.
- ◆ Each private key is known only to the certificate owner.
- ◆ The user has a reliable copy of the certificate authority's public key.

To satisfy these conditions, not only must the user have confidence in the integrity of the certificate authority, but the user must also have obtained the same public key directly from the certificate authority.

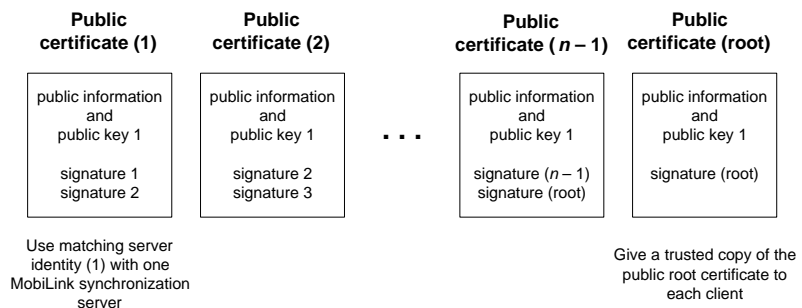
To obtain valid copies of a public key, users of this system typically obtain copies of a self-signed certificate owned by the certificate authority. To foil impostors, the certificate must be obtained by reliable means.

In addition, each client must store the copy of the certificate authority's certificate securely. Should an adversary have access to the user's computer, he or she could replace the certificate authority's certificate with a fake.

Certificate chains

When deploying a replication system, a large number of certificates may be required. The responsibility of signing many certificates may place too great a burden on the certificate authority. To lessen their workload, a certificate authority can delegate signing authority to others. To do so, the certificate authority signs a certificate held by the delegate. The delegate then proceeds to sign certificates using the private key that matches the one in this certificate.

A certificate chain is a sequence of certificates such that each certificate is signed by the next. The final certificate, called the root certificate, is owned by a certificate authority. For example, a server certificate can be signed by a delegate. The delegate's certificate can be signed by a certificate authority. The certificate authority's public key is contained in a third certificate. Such a situation is a chain of three certificates.



In fact, a delegate can also have delegates. Thus, a chain of certificates can be of any length. However, the final certificate is always a self-signed root certificate, owned by a certificate authority.

To trust a chain, a user must trust each of the following:

- ◆ Before signing each certificate, the certificate authority and all delegates made certain that the identity information contained in the certificate correctly identified the certificate owner.
- ◆ Each private key is known only to the certificate owner.
- ◆ The user has a reliable copy of the certificate authority's public key.

All conditions are extremely important. The chain of certificates is only as strong as its weakest link.

Enterprise root certificates

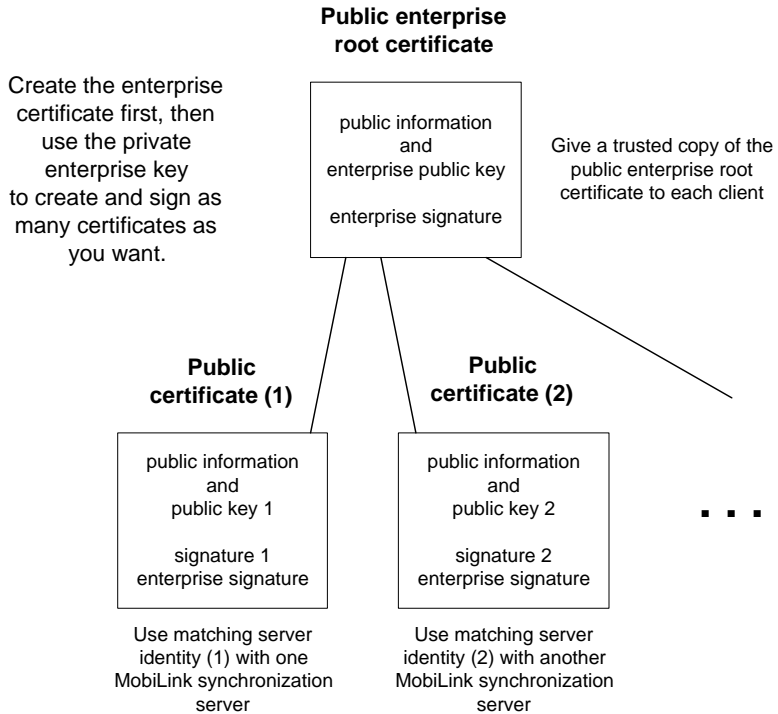
A deployment of MobiLink that involves multiple servers can be improved by assigning each server a unique certificate also signed by a common root certificate. A certificate authority within the enterprise holds the root certificate.

This arrangement has the following advantages:

- ◆ Each MobiLink synchronization server can be given a unique certificate, so that if one site is compromised, the others are not affected.
- ◆ Security is enhanced because the private key for the enterprise root certificate need not be stored on the MobiLink synchronization server.
- ◆ Clients do not need to keep a copy of each server's public certificate, only a copy of the public root certificate because you can configure them to trust any certificate signed by the root certificate.

The security of the system can be improved somewhat by obtaining a globally signed certificate, discussed later, from a commercial certificate authority. In practice, however, this arrangement provides adequate security for many applications.

You can program your clients to verify the values of some certificate fields, as discussed later. In this way, you can ensure that your clients synchronize with particular MobiLink synchronization servers within your organization.



This setup provides more flexibility than self-signed server certificates. For example, you can add a new server and give it a new certificate. If the new certificate is signed with the same enterprise root certificate, existing clients will automatically trust it. Were you, instead, to give each MobiLink synchronization server a self-signed certificate, all clients would require a copy of the new public certificate.

Creating the certificates

The first step in setting up an enterprise-level system is to generate the common self-signed certificate. To generate this root certificate, start `gencert` with the `-r` option.

```
>gencert -r
Certificate Generation Tool
Choose certificate type ((R)SA or (E)CC): E
Generating key pair...
Country: CA
State/Province: Ontario
Locality: Waterloo
Organization: Sybase, Inc.
Organizational Unit: IAS
Common Name: MobiLink
Serial Number: 2003.07.29.02
Certificate valid for how many years: 2
Enter password to protect private key: password2
Enter file path to save certificate: ent_root.crt
Enter file path to save private key: ent_root.pri
Enter file path to save server identity: ent_serv.crt
```

The utility creates three files, which in this example are called *ent_root.crt*, *ent_root.pri*, and *ent_serv.crt*.

- ◆ **ent_root.crt** This file contains the new certificate. This certificate should be published as all clients require a reliable copy.
- ◆ **ent_root.pri** This file contains the private key that matches the public key encoded in the certificate.
- ◆ **ent_serv.crt** This file contains the same information as the above two files, combined. It is intended for use with a MobiLink synchronization server.

The first two of these three files can be used to sign additional, new certificates. To generate a signed certificate, start *gencert* with the *-s* option. Enter the name of the signing certificate file, the name of the signing private-key file, and the password for the signing private key.

```
>gencert -s
Certificate Generation Tool
Choose certificate type ((R)SA or (E)CC): E
Generating key pair...
Country: CA
State/Province: Ontario
Locality: Waterloo
Organization: Sybase, Inc.
Organizational Unit: IAS
Common Name: MobiLink
Serial Number: 2003.07.29.03
Certificate valid for how many years: 1
Enter file path of signer's certificate: ent_root.crt
Enter file path of signer's private key: ent_root.pri
Enter password for signer's private key: password2
Enter password to protect private key: password3
Enter file path to save server identity: serv1.crt
```

This time, gencert creates only one file. This file contains the signed certificate and the private key. It is intended for use with a MobiLink synchronization server.

Repeat this last step as many times as necessary to create a signed certificate for each MobiLink synchronization server.

```
>gencert -s
Certificate Generation Tool
Choose certificate type ((R)SA or (E)CC): E
Generating key pair...
Country: CA
State/Province: Ontario
Locality: Waterloo
Organization: Sybase, Inc.
Organizational Unit: IAS
Common Name: MobiLink
Serial Number: 2003.07.29.04
Certificate valid for how many years: 2
Enter file path of signer's certificate: ent_root.crt
Enter file path of signer's private key: ent_root.pri
Enter password for signer's private key: password2
Enter password to protect private key: password4
Enter file path to save server identity: serv2.crt
```

You now have the following files:

- ◆ **ent_root.crt** The root certificate.
- ◆ **ent_root.pri** The root private key.
- ◆ **ent_serv.crt** The root combined certificate.
- ◆ **serv1.crt** The combined certificate for the first MobiLink synchronization server.
- ◆ **serv2.crt** The combined certificate for the second MobiLink synchronization server.

You do not need the combined root certificate because no MobiLink synchronization server uses it directly. Instead, you created a separate certificate for each MobiLink synchronization server.

Using the signed certificates

You can use the signed certificates for server-authentication by following these steps:

1. Supply a copy of the public root certificate to all clients. When the client first contacts the MobiLink synchronization server, the server sends the client a copy of its own public certificate. This certificate bears the

signature of the root certificate. The client can detect fake certificates by verifying that the root signature matches the public key in their copy of the root certificate.

2. Tell each client that it is to trust only servers whose certificates bear the signature of the root certificate. For Adaptive Server Anywhere clients, use the `trusted_certificates` security parameter. For example, you can tell an Adaptive Server Anywhere client to trust only the `ent_cert.crt` certificate by including this parameter in the address clause of the synchronization subscription, as in the following example.

```
CREATE SYNCHRONIZATION SUBSCRIPTION
FOR 'user001' TO test
ADDRESS 'host=myhost;security=ecc_tls (
trusted_certificates=ent_cert.crt)'
```

To tell an UltraLite client to trust only the desired certificate, name the trusted certificate using the `-r` option when running the UltraLite generator, as follows. Open a command prompt and run the following command line:

```
ulgen -c "dsn=UltraLite 9.0 Sample;uid=DBA;pwd=SQL"
-r ent_cert.crt -j custapi
```

3. When you start each MobiLink synchronization server, specify the name of that server's certificate file and the corresponding password. Enter each command on one line.

```
dbmlsrv9 -c "dsn=UltraLite 9.0 Sample;uid=DBA;pwd=SQL"
-x tcpip ( port=3333;
security=ecc_tls ( certificate=serv1.crt;
certificate_password=password3 ) )
dbmlsrv9 -c "dsn=UltraLite 9.0 Sample;uid=DBA;pwd=SQL"
-x tcpip ( port=4444;
security=ecc_tls ( certificate=serv2.crt;
certificate_password=password4 ) )
```

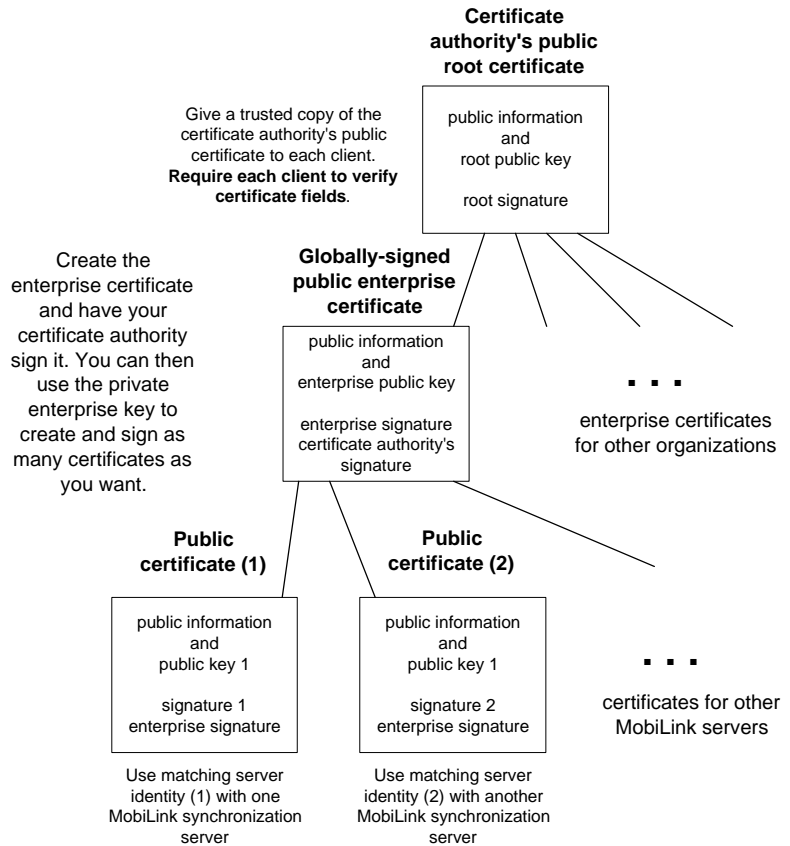
Globally signed certificates

You can improve the security of a multi-server MobiLink setup by assigning each server a unique certificate that is signed by a common root certificate. You can improve it further using a certificate signed by a commercial certificate authority. Such a certificate is called a global certificate or a globally-signed certificate. A commercial certificate authority is an organization that is in the business of creating high-quality certificates and using these certificates to sign other certificates.

A global certificate has the following advantages:

- ◆ Security requires that both parties trust the root certificate. In the case of inter-company communication, common trust in an outside, recognized authority may increase confidence in the security of the system because a certificate authority must guarantee the accuracy of the identification information in any certificate that it signs.
- ◆ Security is enhanced when keys are created using pseudo-random data of high quality. The data used with the gencert utility is of cryptographic quality, but other, even better methods can be used in controlled environments.
- ◆ The private key for the root certificate must remain private. An enterprise may not have a suitable place to store this crucial information, whereas a certificate authority can afford to design and maintain dedicated facilities.

When using a globally signed certificate, each client must verify certificate field values to avoid trusting certificates that the same certificate authority has signed for other clients. This process is described in the next section.



Obtaining server-authentication certificates

MobiLink transport-layer security is based on Certicom SSL/TLS Plus libraries, which require elliptic-curve or RSA certificates. You can obtain a global certificate from any certificate authority that can supply certificates in the correct format. Two such companies are VeriSign and Entrust Technologies.

☞ For more information, see <http://www.verisign.com/> or http://www.entrust.com/certificate_services/index.htm.

☞ There are several ways to obtain certificates. One way is to use the Certicom reqtool utility, which is installed when you install the security component. This tool creates a server certificate and a global certificate request. Copy the contents of the public certificate onto your clipboard, and paste them into the form on the Web site of the certificate-issuing authority. Only submit the public component of the certificate request. You must not disclose your private key.

For more information about this procedure, see the document *reqtool.pdf*, located in the *win32* subdirectory of your SQL Anywhere 9 installation. It is installed when you install the security component.

Example

The following example creates an elliptic-curve certificate:

```
> reqtool
-- Certicom Corp. Certificate Request Tool 3.0d1 --
Choose certificate request type:
  E - Personal email certificate request.
  S - Server certificate request.
  Q - Quit.
Please enter your request [Q] : S
Choose key type:
  R - RSA key pair.
  D - DSA key pair.
  E - ECC key pair.
  Q - Quit.
Please enter your request [Q] : E
Using curve ecl63a02. Generating key pair (please wait)...
Country: CA
State: Ontario
Locality: Waterloo
Organization: Sybase, Inc.
Organizational Unit: IAS
Common Name: MobiLink
Enter password to protect private key : password5
Enter file path to save request : global.req
Enter file path to save private key : global.pri
```

The file *global.req* contains the public certificate and request information. Paste the contents of this file into the form on the certificate-issuing Web site.

The file *global.pri* contains the private key for the enterprise certificate. This file is protected by the password you entered, but since the protection provided by the password is weak, you must store this file in a secure location.

Using a global certificate as a server certificate

You can use your global certificate directly as a MobiLink synchronization server certificate. To do so, you must create a server identity certificate by concatenating the public and private certificates. Open a command prompt and run the following command line:

```
copy global.crt+global.pri global2.crt
```

You can now start a MobiLink synchronization server, specifying the new certificate and the password for your private certificate. Open a command prompt and run the following command line:

```
dbmlsrv9 -c "dsn=UltraLite 9.0 Sample;uid=DBA;pwd=SQL" -x tcpip
( security=ecc_tls( certificate=global2.crt;
certificate_password=password5 ) )
```

You can hide the contents of the command line using the File Hiding utility, *dbfhide*. For more information, see “The File Hiding utility” [*ASA Database Administration Guide*, page 466].

You must also ensure that clients contacting your MobiLink synchronization server trust the certificate. To do so, you must tell the clients to trust the root certificate in the chain. In this case, the root certificate in the chain is a certificate held by the certificate authority.

By default, MobiLink clients trust certificates signed by the Sybase root certificate used to sign the sample certificate included with MobiLink.

For better security, however, you should ensure that clients consider only the root certificate of your certificate authority to be valid.

You can tell an Adaptive Server Anywhere MobiLink client to accept only a particular root certificate by naming only this certificate in the Address clause of the SQL CREATE SYNCHRONIZATION SUBSCRIPTION statement. For example, to trust certificates from XXX:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
FOR 'user001' TO test -pub
ADDRESS 'host=myhost;security=ecc_tls (
trusted_certificates=XXX.crt )'
```

To tell an UltraLite client to trust only the XXX root certificate, name the trusted certificate using the *-r* option when running the UltraLite generator,

as follows. Open a command prompt and run the following command line:

```
> ulgen -c "dsn=UltraLite 9.0 Sample;uid=DBA;pwd=SQL"  
-r XXX.crt -j custapi
```

Verifying certificate fields

Global certificates have one potentially serious flaw. Because the MobiLink clients, as configured above, trust all certificates signed by the certificate authority, they may also trust certificates that the same certificate authority has issued to other companies. Without a means to discriminate, your clients might mistake a competitor's MobiLink synchronization server for your own and accidentally send it sensitive information.

Similar precautions can be required in other scenarios. A company may use an enterprise certificate, but it may still be important to verify with which department a MobiLink client is connected.

This problem can be resolved by requiring your clients to test the value of fields in the identity portion of the certificate. Three fields in the certificate can be verified. You can verify any or all of the following three fields:

- ◆ Organization
- ◆ Organizational Unit
- ◆ Common Name

To verify the fields, you supply the acceptable value. For example, the following SQL statement tells an Adaptive Server Anywhere client to check all three fields and to accept only the named values:

```
CREATE SYNCHRONIZATION SUBSCRIPTION
FOR 'user01'
TO test
ADDRESS 'port=3333;security=ecc_tls(      trusted_
      certificates=certicom.crt;
      certificate_company=Sybase, Inc.;
      certificate_unit=iAnywhere;certificate_name=sample )'
```

You can verify the fields from an UltraLite client in a similar manner. The precise syntax depends upon the interface used to build the application. The following fragment of C code accomplishes the same task when developing the UltraLite application using embedded SQL in C or C++:

```
ul_synch_info info;
. . .
info.security_parms =
    UL_TEXT ( "certificate_company=Sybase, Inc." )
    UL_TEXT ( ";" )
    UL_TEXT ( "certificate_unit=iAnywhere" )
    UL_TEXT ( ";" )
    UL_TEXT ( "certificate_name=sample" );
. . .
ULSynchronize( &info );
```

This example verifies all three fields. You can instead choose to verify only one or two fields.

Verifying fields in certificate chains

When the first certificate is part of a chain, all the specified field values are checked in that certificate. If specified, the company name is also checked in all the other certificates, except for the root certificate. This arrangement allows for the case that the root certificate is held by a certificate authority. In this case, the field values of the root certificate will be different, as it is owned by the certificate authority, rather than your company or organization.

Using a globally-signed certificate as an enterprise certificate

Instead of using a global certificate as a server certificate, it is possible to instead use it to sign other certificates, as you would an enterprise certificate. This setup lets you combine the benefits of a global certificate and an enterprise certificate. The most important advantage is that you need not store the private key for your global certificate on the computer running the MobiLink synchronization server.

To create such a setup, generate a unique certificate for each MobiLink synchronization server. When you do so, sign them with your global certificate.

The following example displays how two server certificates can be generated and signed by the global certificate:

```
>gencert -s
Certificate Generation Tool
Choose certificate type ((R)SA or (E)CC): E
Generating key pair...
Country: CA
State/Province: Ontario
Locality: Waterloo
Organization: Sybase
Organizational Unit: IAS
Common Name: MobiLink
Serial Number: 2003.07.29.06
Certificate valid for how many years: 1
Enter file path of signer's certificate: global.crt
Enter file path of signer's private key: global.pri
Enter password for signer's private key: password5
Enter password to protect private key: password6
Enter file path to save server identity: serv6.crt
>gencert -s
Certificate Generation Tool
Choose certificate type ((R)SA or (E)CC): E
Generating key pair...
Country: CA
State/Province: Ontario
Locality: Waterloo
Organization: Sybase
Organizational Unit: IAS
Common Name: MobiLink
Serial Number: 2003.07.29.07
Certificate valid for how many years: 1
Enter file path of signer's certificate: global.crt
Enter file path of signer's private key: global.pri
Enter password for signer's private key: password5
Enter password to protect private key: password7
Enter file path to save server identity: serv7.crt
```

The above commands generate two server identity certificates, intended for use with two MobiLink synchronization servers.

- ◆ **serv6.crt** The server identity certificate for MobiLink synchronization server #1.
- ◆ **serv7.crt** The server identity certificate for MobiLink synchronization server #2.

Both certificates are signed by *global.crt*, which in turn is signed by your certificate authority's root certificate.

You can start these two MobiLink synchronization servers with the following commands, entered one command per line.

```
dbmlsrv9 -c "dsn=UltraLite 9.0 Sample;uid=DBA;pwd=SQL" -x tcpip
( port=3333;security=ecc_tls ( certificate=serv6.crt;
certificate_password=password6 ) )
dbmlsrv9 -c "dsn=UltraLite 9.0 Sample;uid=DBA;pwd=SQL" -x tcpip
( port=4444;security=ecc_tls ( certificate=serv7.crt;
certificate_password=password7 ) )
```

You can hide the contents of the command line using the File Hiding utility, dbfhide. For more information, see “The File Hiding utility” [*ASA Database Administration Guide*, page 466].

In addition, you must ensure that each client trusts your certificate authority’s root certificate.

PART II

MOBILINK TUTORIALS

This part provides hands-on tutorials that introduce you to the basic techniques of creating MobiLink synchronization systems.

CHAPTER 17

Tutorial: Synchronizing Adaptive Server Anywhere Databases

About this chapter

This chapter provides a tutorial to guide you through the process of setting up a synchronization system when the consolidated and remote databases are both Adaptive Server Anywhere databases.

Contents

Topic:	page
Introduction	370
Lesson 1: Creating and populating your databases	371
Lesson 2: Running the MobiLink synchronization server	375
Lesson 3: Running the MobiLink synchronization client	377
Tutorial cleanup	379
Summary	380
Further reading	381

Introduction

In this tutorial, you create a consolidated database and a remote database. You then synchronize these databases using MobiLink synchronization technology.

Timing

The tutorial takes about 30 minutes.

Competencies and experience

You will require:

- ◆ Knowledge of and/or experience with command line processing.
- ◆ Knowledge of and/or experience with Interactive SQL.
 - For more information, see “Using Interactive SQL” [*ASA Getting Started*, page 67].

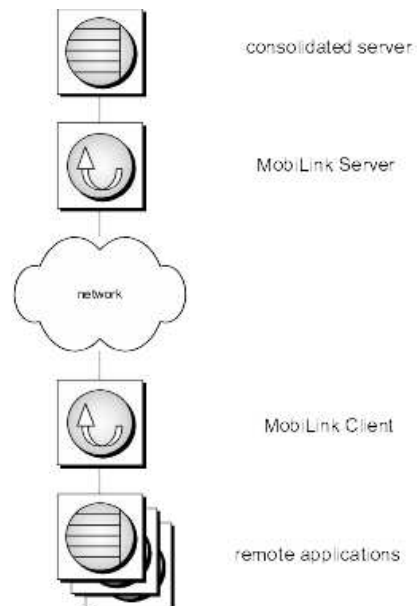
Goals

You will gain competence and familiarity with:

- ◆ The MobiLink synchronization server and client as an integrated system.
- ◆ The MobiLink synchronization server and client command lines and options.

Key concepts

The MobiLink synchronization server connects to the consolidated database using ODBC. The MobiLink synchronization client connects to your remote database. The MobiLink synchronization server and client function as a group, managing the upload and download of data from one database to another, as shown in the figure below.



Lesson 1: Creating and populating your databases

MobiLink synchronization requires that you have data in a relational database, an ODBC data source for each database, and two compatible databases.

Create your database files

The first step is to create each of the databases. In this procedure, you build a consolidated database and a remote database using the *dbinit* utility from a command line.

The *dbinit* utility creates a database file with no user tables or procedures. You create your database schema when you define, within the newly-initialized file, user-defined tables and procedures.

❖ To create your database files

1. Open a command prompt and navigate to the *Samples\MobiLink\AutoScripting* subdirectory of your SQL Anywhere 9 installation.
2. Create a consolidated database for this tutorial. Run the following command line:

```
dbinit consol.db
```

If this tutorial has been previously run on your computer, *consol.db* and *consol.log* may already exist. This will cause *dbinit* to fail. Delete these files before running *dbinit*.

3. Create the remote database for this tutorial. Run the following command line:

```
dbinit remote.db
```

If this tutorial has been previously run on your computer, *remote.db* and *remote.log* may already exist. This will cause *dbinit* to fail. Delete these files before running *dbinit*.

4. Verify the successful creation of these database files by listing the contents of the directory. You should see *consol.db* and *remote.db* in the listing.

Create ODBC data sources

☞ You are now ready to build ODBC data sources through which you can connect to your Adaptive Server Anywhere databases.

☞ For more information about creating ODBC data sources, see “The Data Source utility” [*ASA Database Administration Guide*, page 472].

❖ To create ODBC data sources

1. Open a command prompt and navigate to the *Samples\MobiLink\AutoScripting* subdirectory of your SQL Anywhere 9 installation.
2. Create your ODBC data source for a consolidated database by running the following command line:

```
dbdsn -w test_consol -y -c  
      "uid=DBA;pwd=SQL;dbf=consol.db;eng=Consol"
```

This command line specifies the following options:

- ◆ **-w** Creates a data source definition.
- ◆ **-y** Delete or overwrite data source without confirmation.
- ◆ **-c** Specifies the connection parameters as a connection string.
☞ For more information, see “Data Source utility options” [ASA *Database Administration Guide*, page 475].

3. Create an ODBC data source for a remote database by running the following command line:

```
dbdsn -w test_remote -y -c  
      "uid=DBA;pwd=SQL;dbf=remote.db;eng=Remote"
```

4. Verify the successful creation of your data sources as follows.

❖ To verify your new data sources

1. Choose Start ► Programs ► Sybase SQL Anywhere 9 ► Adaptive Server Anywhere ► ODBC Administrator.
The ODBC Data Source Administrator appears.
2. Click the User DSN tab.
3. Scroll through the list to find the test_remote and test_consol data sources.
4. Select each data source and click Configure.
5. Test your data source by clicking the Test Connection button.
6. Click OK to close the ODBC Data Source Administrator.

Create your schema

The following procedure executes SQL statements using the Interactive SQL utility to create and populate tables in the consolidated database. It also creates tables and inserts synchronization subscriptions and publications into the remote database.

The SQL files, *build_consol.sql* and *build_remote.sql*, are created specifically for this tutorial.

❖ To call and run your scripts using Interactive SQL

1. Open a command prompt and navigate to the *Samples\MobiLink\AutoScripting* subdirectory of your SQL Anywhere 9 installation.

2. Run the following command line:

```
dbisql -c "dsn=test_consol;astop=no" build_consol.sql
```

The SQL statements in *build_consol.sql* create and populate the emp and cust tables in the consolidated database.

This step includes `astop=no` to instruct the server not to shut down when the *dbisql* utility shuts down.

3. Run the following command line:

```
dbisql -c "dsn=test_remote;astop=no" build_remote.sql
```

The SQL statements in *build_remote.sql* create the remote tables emp and cust, and insert synchronization subscriptions and publications.

4. Verify the creation of the emp and cust tables in the remote and consolidated databases using Interactive SQL.
 - ◆ Open Interactive SQL by typing *dbisql* at a command prompt. Connect using the test_consol DSN as DBA, using SQL as the password.
 - ◆ Execute the following SQL statement by typing it into the SQL Statements pane and pressing F9.

```
SELECT * FROM emp, cust
```

The tables in the consolidated database are populated with data.

- ◆ Connect using the test_remote DSN and execute the following SQL statement:

```
SELECT * FROM emp, cust
```

The tables in the remote database are empty.

5. Leave the consolidated and remote databases running for the next lesson.

Further reading

☞ For more information about creating remote databases, see [“Creating a remote database” on page 168](#).

☞ For more information about creating subscriptions and publications, see [“Publishing data” on page 171](#).


☞ For more information about creating databases, see [“The Initialization utility” \[ASA Database Administration Guide, page 485\]](#) and [“Creating a](#)

database using the dbinit command-line utility” [*ASA Database Administration Guide*, page 486].

☞ For more information about running Interactive SQL, see “The Interactive SQL utility” [*ASA Database Administration Guide*, page 492] and “Using Interactive SQL” [*ASA Getting Started*, page 67].

☞ For more information about SELECT statements, see “SELECT statement” [*ASA SQL Reference*, page 541].

Lesson 2: Running the MobiLink synchronization server

 Your consolidated database must be running prior to running MobiLink. If you shut down your consolidated database following Lesson 1, you should restart the database. You can start the MobiLink synchronization server from a command prompt.

❖ To start the MobiLink synchronization server

1. Open a command prompt and navigate to the *Samples\MobiLink\AutoScripting* subdirectory of your SQL Anywhere 9 installation.
2. Run the following command line:

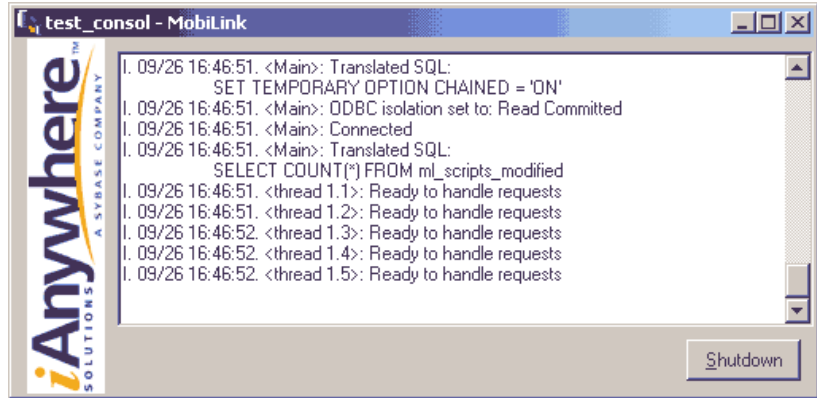
```
dbmlsrv9 -c "dsn=test_consol" -o mlserver.mls -v+ -dl -za -zu+
```

This command line specifies the following options:

- ◆ **-c** The connection string for the MobiLink synchronization server uses the DSN for the consolidated database. For more information, see “-c option” [*MobiLink Synchronization Reference*, page 10].
- ◆ **-o** The `-o` option is used to specify the message log file. For more information, see “-o option” [*MobiLink Synchronization Reference*, page 13].
- ◆ **-v+** The `-v+` option sets verbose logging on. For more information, see “-v option” [*MobiLink Synchronization Reference*, page 21].
- ◆ **-dl** The `-dl` option sets the display log feature ON. For more information, see “-dl option” [*MobiLink Synchronization Reference*, page 12].
- ◆ **-za** The `-za` option turns automated scripting ON. For more information, see “-za option” [*MobiLink Synchronization Reference*, page 28].
- ◆ **-zu+** The `-zu+` option automates the user authentication process. For more information, see “-zu option” [*MobiLink Synchronization Reference*, page 31].

The options `-o`, `-v`, and `-dl` are chosen to provide debugging and troubleshooting information. Using these options in a production environment may affect performance. They typically are not used in a production environment.

Once you have executed the MobiLink synchronization server command, the output below appears.



If MobiLink is already running when you attempt to run *dbmlsrv9*, you will receive an error message. Shut down the current instance of MobiLink and run the command again.

Further reading

➞ For more information about the MobiLink synchronization server, see [“The MobiLink synchronization server” on page 16](#).

➞ For a complete list of *dbmlsrv9* options, see “MobiLink Synchronization Server Options” [*MobiLink Synchronization Reference*, page 3].

Lesson 3: Running the MobiLink synchronization client

Adaptive Server Anywhere clients initiate MobiLink synchronization by using the dbmlsync utility.

❖ To start the MobiLink synchronization client

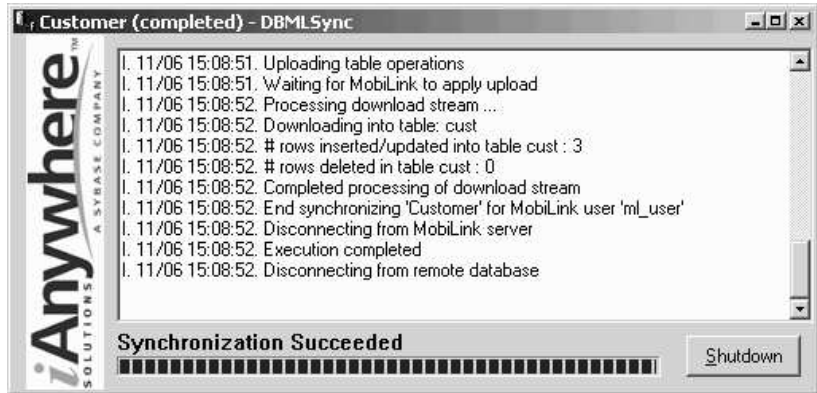
1. Open a command prompt and navigate to the *Samples\MobiLink\AutoScripting* subdirectory of your SQL Anywhere 9 installation.
2. Run the following command line:

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

This command line specifies the following options:

- ◆ **-c** Supply database connection parameters. For more information, see “-c option” [*MobiLink Synchronization Reference*, page 42].
- ◆ **-o** Specify the message log file. For more information, see “-o option” [*MobiLink Synchronization Reference*, page 74].
- ◆ **-v** Verbose operation. For more information, see “-v option” [*MobiLink Synchronization Reference*, page 80].
- ◆ **-e** Extended options. Specifying “SendColumnNames=ON” sends column names to MobiLink. This is required when you use -za in the dbmlsrv9 command line. For more information, see “SendColumnNames (scn) extended option” [*MobiLink Synchronization Reference*, page 62].

Once you have executed the MobiLink synchronization client command, the output below appears to indicate that synchronization has succeeded. After synchronization, the remote database is populated with the data from the consolidated database.



Further reading

For more information about dbmsync options, see “MobiLink synchronization client” [*MobiLink Synchronization Reference*, page 36].

➡ For more information about remote clients, see [“MobiLink clients” on page 19](#).

➡ For more information about dbmsync, see [“Initiating synchronization” on page 185](#).

Tutorial cleanup

You should remove tutorial materials from your computer.

❖ To remove tutorial materials from your computer

1. Close the Adaptive Server Anywhere, MobiLink, and synchronization client windows by right-clicking each taskbar item and choosing Close.
2. Delete all tutorial-related data sources.
 - ◆ Choose Start ► Programs ► Sybase SQL Anywhere 9 ► Adaptive Server Anywhere ► ODBC Administrator.
The ODBC Data Source Administrator appears.
 - ◆ Select `test_remote` and `test_consol` from the list of User Data Sources. Click Remove.
 - ◆ Click OK to close the ODBC Data Source Administrator.
3. Delete the consolidated and remote databases.
 - ◆ Open Windows Explorer and navigate to the *Samples\MobiLink\AutoScripting* subdirectory of your SQL Anywhere 9 installation.
 - ◆ Delete *remote.db*, *remote.log*, *consol.db*, and *consol.log*.

Summary

During this tutorial, you:

- ◆ Created and populated Adaptive Server Anywhere consolidated and remote databases.
- ◆ Started a MobiLink synchronization server.
- ◆ Started the MobiLink synchronization client and synchronized the remote database with the consolidated database.

Learning
accomplishments

During this tutorial, you gained:

- ◆ Familiarity with the MobiLink synchronization server and client as an integrated system.
- ◆ Competence in executing MobiLink synchronization server and client commands.
- ◆ Familiarity with the MobiLink synchronization server and client command lines and options.

Further reading

The following documentation sections are good starting points for further reading:

- ☞ For more information about Adaptive Server Anywhere remote databases, see [“Adaptive Server Anywhere Clients” on page 167](#).
- ☞ For more information about running the MobiLink synchronization server, see [“Synchronization Basics” on page 7](#).
- ☞ For more information about synchronization scripting, see [“Writing Synchronization Scripts” on page 37](#).

CHAPTER 18

Tutorial: Writing SQL Scripts Using Sybase Central

About this chapter

This chapter provides a tutorial to guide you through the process of setting up a synchronization system when the consolidated and remote databases are both Adaptive Server Anywhere databases.

This tutorial guides you through the process of creating and modifying synchronization scripts. You also view MobiLink objects using the Sybase Central MobiLink plug-in.

Contents

Topic:	page
Introduction	384
Lesson 1: Creating your databases	385
Lesson 2: Creating scripts for your synchronization	389
Lesson 3: Running the MobiLink synchronization server	392
Lesson 4: Running the MobiLink synchronization client	394
Lesson 5: Monitoring your MobiLink synchronization using log files	396
Tutorial cleanup	398
Further reading	399

Introduction

In this tutorial, you create a consolidated database and a remote database, and write scripts to perform synchronization. You then synchronize the two databases.

Timing

The tutorial takes about 50 minutes.

Competencies and experience

You will require:

- ◆ An understanding of the role of synchronization scripts in the synchronization process.
- ◆ An understanding of the properties of publications and subscriptions.

Goals

The goals for the tutorial are to gain competence and familiarity with the following tasks:

- ◆ Creating MobiLink synchronization scripts using Sybase Central.
- ◆ Troubleshooting MobiLink errors.

Key concepts

The key concepts you will learn from this tutorial include:

- ◆ The role of the MobiLink synchronization server and client in the synchronization process.
- ◆ How to create an ODBC connection and set the properties of an ODBC connection for Adaptive Server Anywhere.
- ◆ How to create synchronization scripts to work with MobiLink.
- ◆ How to initialize a database.

Suggested background reading

☞ For more information about Sybase Central, see “Tutorial: Managing Databases with Sybase Central” [*Introducing SQL Anywhere Studio*, page 47].

☞ For more information about synchronization, see “[Tutorial: Synchronizing Adaptive Server Anywhere Databases](#)” on page 369.

☞ For more information about synchronization scripts, see “[Introduction to synchronization scripts](#)” on page 38.

Lesson 1: Creating your databases

MobiLink synchronization requires that you have data in a relational database, an ODBC data source for the consolidated database, and two supported databases.

Before you start, to ensure that there are no conflicts with databases or DSNs created during other tutorials, navigate to the *Samples\MobiLink\Autoscripting* subdirectory of your SQL Anywhere 9 installation and run *clean.bat*.

Create databases

In this procedure, you build a consolidated database and a remote database using Sybase Central.

❖ To create your databases

1. Choose Start ► Programs ► Sybase SQL Anywhere 9 ► Sybase Central. Sybase Central appears.
2. In Sybase Central, choose Tools ► Adaptive Server Anywhere 9 ► Create Database.
The Create a Database wizard appears. Click Next.
3. Leave the default of Create a Database on this Computer. Click Next.
4. Enter the following filename and path for the database:
Samples\MobiLink\Autoscripting\test_consol.db
Click Finish.
5. Repeat steps 2 through 4 for the remote database, using the filename *test_remote.db* in place of *test_consol.db*.

Create ODBC data sources

☞ You will now build ODBC data sources through which you can connect to your Adaptive Server Anywhere 9 databases.

❖ To create ODBC data sources

1. In Sybase Central, choose Tools ► Adaptive Server Anywhere 9 ► Open ODBC Administrator.
2. Create your ODBC data source for a database:
 - ◆ Click the User DSN tab and click Add.
The Create New Data Source dialog appears.
 - ◆ Select Adaptive Server Anywhere 9.0 and click Finish.
The ODBC Configuration for Adaptive Server Anywhere 9.0 dialog appears.

-
3. Click the ODBC tab and enter **test_consol** as the Data Source Name.
 4. Click the Login tab and enter the User ID **dba** and password **SQL**.
 5. Enter **test_consol** under Server Name.
 6. Click the Database tab. Click Browse to locate *test_consol.db*.
 7. Clear the Stop Database After Last Disconnect checkbox.
 8. Click OK to return to the ODBC Data Source Administrator.
 9. Repeat steps 2 through 8 using the name **test_remote** instead of **test_consol** for the ODBC connection name, database file name, and server name.
 10. Click OK to close the ODBC Data Source Administrator.

❖ **To verify your new data sources**

1. In Sybase Central, choose Tools ► Adaptive Server Anywhere 9 ► Open ODBC Administrator.
The ODBC Data Source Administrator appears.
2. Click the User DSN tab.
3. Scroll through the list to find *test_consol* and *test_remote*.
4. Select each data source and click Configure.
5. Test your data source by clicking the Test Connection button.

Create your schema

The following procedure executes SQL statements using the Interactive SQL utility to create and populate tables in the consolidated database. It also creates tables and inserts synchronization subscriptions and publications into the remote database.

The SQL files *build_consol.sql* and *build_remote.sql* are created specifically for this tutorial.

❖ **To run your scripts in Interactive SQL**

1. In Sybase Central, choose Tools ► Adaptive Server Anywhere 9 ► Open Interactive SQL.
2. Connect to the consolidated database using the `test_consol` ODBC data source.
3. Choose File ► Run Script.

The Open File dialog appears. Browse to *Samples\MobiLink\AutoScripting\build_consol.sql*.

The SQL statements create and populate two tables, `emp` and `dept`.

4. Verify the successful creation of each of the `emp` and `cust` tables.
 - ◆ Execute the following command in the SQL Statements pane:

```
SELECT * FROM emp, cust
```

The tables should be populated with data.

5. Close Interactive SQL.
6. Repeat steps 1 to 3 for the remote, using the ODBC data source *test_remote* and the file *build_remote.sql*.

The SQL statements create, but do not populate, two tables, `emp` and `dept`. Synchronization subscriptions and publications define the synchronization parameters for the MobiLink synchronization client.

7. Verify the successful creation of each of the `emp` and `cust` tables.
 - ◆ Execute the following command in the SQL Statements pane:

```
SELECT * FROM emp, cust
```

The tables should contain no data.

8. Close Interactive SQL. Leave the consolidated and remote databases running.

Synchronization
subscriptions and
publications

☞ MobiLink synchronization requires a MobiLink synchronization subscription and publication. Synchronization subscriptions and publications are stored in the remote database.

Publications identify the tables and columns on your remote database that you want synchronized. These tables and columns are called articles. After you create articles for your publication, the MobiLink synchronization server uses the information contained in the publication to construct SQL statements that are used to transfer data from your remote to your consolidated database.

This tutorial uses the SQL file *build_remote.sql*. The SQL statements in *build_remote* perform the following steps to a build publication and a synchronization subscription:

- ◆ The following SQL statement creates the emp_cust synchronization publication that identifies two articles, the cust and emp tables:

```
CREATE PUBLICATION emp_cust (TABLE cust, TABLE emp)
```

- ◆ The following SQL statement creates a synchronization user, ml_user:

```
CREATE SYNCHRONIZATION SUBSCRIPTION USER ml_user
```

- ◆ The following SQL statement creates a synchronization subscription for ml_user to the emp_cust publication. The synchronization subscription identifies a MobiLink user, the publication name (emp_cust), and optional communication parameters.

```
CREATE SYNCHRONIZATION SUBSCRIPTION TO emp_cust FOR ml_user
```

- ◆ The following SQL statement specifies the communication mode for the synchronization subscription as TCP/IP:

```
ALTER SYNCHRONIZATION SUBSCRIPTION  
TO emp_cust FOR ml_user  
TYPE TCPIP ADDRESS 'host=localhost'
```

In the next lesson, you will learn how to modify these objects in Sybase Central.

Further reading

☞ For more information about consolidated databases, see [“Consolidated database” on page 10](#).

☞ For more information about remote databases, see [“MobiLink clients” on page 19](#).

☞ For more information about Interactive SQL, see “The Interactive SQL utility” [ASA Database Administration Guide, page 492].

☞ For more information about creating ODBC data sources, see “The Data Source utility” [ASA Database Administration Guide, page 472].

☞ For more information about defining publications and subscriptions, see [“Publishing data” on page 171](#).

Lesson 2: Creating scripts for your synchronization

You can view, write, and modify synchronization scripts using Sybase Central. In this section you add scripts to the consolidated database.

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

❖ To add a script version

1. Start Sybase Central and connect to the test_consol database using the MobiLink plug-in.
2. Select the Versions folder. Double-click Add Version.
The Add a New Script Version dialog appears.
3. Name the new version **default**. Click Finish.

❖ To add synchronized tables to your consolidated database

1. In the MobiLink Synchronization plug-in of Sybase Central, select the Tables folder and double-click DBA.
You will see two tables, emp and cust.
2. Right-click each table and choose Add to Synchronized Tables.

Now that you have designated these tables as synchronized, you can add a new table script for each upload and download to the consolidated database.

❖ **To add table scripts to each synchronized table**

1. In the MobiLink Synchronization plug-in of Sybase Central, select the Synchronized Tables folder. You will see two tables, emp and cust. Double-click the emp table.
2. Double-click Add Table Script. The following dialog appears.



3. Select the **upload_insert** event from the dropdown list.
4. Click Finish.
5. Type the following SQL statement into the edit screen:

```
INSERT INTO emp (emp_id, emp_name)
VALUES ( ?, ? )
```
6. Save the script.
7. Close the dialog.
8. Repeat steps 2 to 7 for the **download_cursor** event using the following SQL statement:

```
SELECT emp_id, emp_name FROM emp
```
9. Select the cust table.
10. Repeat steps 2 to 7 for the **upload_insert** event using the following SQL statement:

```
INSERT INTO cust
(cust_id, emp_id, cust_name)
VALUES ( ?, ?, ? )
```

11. Repeat steps 2 to 7 for the **download_cursor** event using the following SQL statement:

```
SELECT cust_id, emp_id, cust_name
FROM cust
```

Further reading

☞ For more information about the scripts you just created, see “upload_insert table event” [*MobiLink Synchronization Reference*, page 218] and “download_cursor cursor event” [*MobiLink Synchronization Reference*, page 133].

☞ For more information about script versions, see [“Script versions” on page 49](#).

☞ For more information about adding scripts, see [“Adding and deleting scripts in your consolidated database” on page 51](#).

☞ For more information about writing table scripts, see [“Table scripts” on page 46](#).

For more information about writing synchronization scripts, see [“Writing Synchronization Scripts” on page 37](#).

Lesson 3: Running the MobiLink synchronization server

☞ The MobiLink synchronization server can be started from a command prompt. Since the MobiLink synchronization server is a client to the consolidated database, your consolidated database must be started prior to starting MobiLink. If you shut down your consolidated database following Lesson 1, you should restart the database.

❖ To start the MobiLink synchronization server

1. Ensure that your consolidated database is running by looking in the system tray for the SQL icon.
2. Open a command prompt and navigate to *Samples\MobiLink\Autoscripting*. Run the following command:

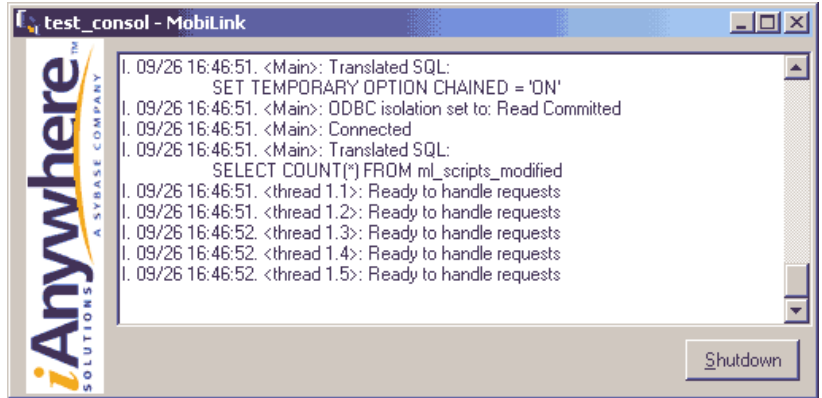
```
dbmlsrv9 -c "dsn=test_consol" -o mlserver.mls -v+ -dl -zu+
```

This command line specifies the following options:

- ◆ **-c** Supply database connection parameters. For more information, see “-c option” [*MobiLink Synchronization Reference*, page 10].
- ◆ **-o** Specify the message log file. For more information, see “-o option” [*MobiLink Synchronization Reference*, page 13].
- ◆ **-v+** Sets verbose logging on. For more information, see “-v option” [*MobiLink Synchronization Reference*, page 21].
- ◆ **-dl** Sets the display log feature ON. For more information, see “-dl option” [*MobiLink Synchronization Reference*, page 12].
- ◆ **-zu+** Automates the user authentication process. For more information, see “-zu option” [*MobiLink Synchronization Reference*, page 31].

The options -o, -v, and -dl are chosen to provide debugging and troubleshooting information. Using these options in a production environment may affect performance. They typically are not used in a production environment.

Once you have executed the MobiLink synchronization server command, the output below appears.



If MobiLink is already running when you attempt to run dbmlsrv9, you will receive an error message. Shut down the current instance of MobiLink and attempt to run the command again.

Further reading

For more information about dbmlsrv9, see [“The MobiLink synchronization server” on page 16](#).

Lesson 4: Running the MobiLink synchronization client

The MobiLink synchronization client may now be started from a command prompt. Adaptive Server Anywhere clients initiate MobiLink synchronization by using the dbmlsync utility.

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

❖ To start the MobiLink client

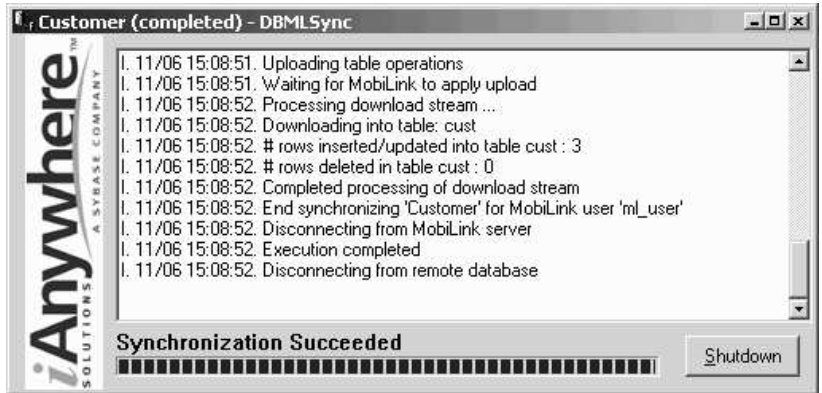
1. Ensure the MobiLink synchronization server is running by looking for the MobiLink icon in the system tray.
2. Open a command prompt and navigate to the *Samples\MobiLink\AutoScripting* subdirectory of your SQL Anywhere 9 installation.
3. Run the following command line:

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

This command line specifies the following options:

- ◆ **-c** Supply database connection parameters. For more information, see “-c option” [*MobiLink Synchronization Reference*, page 10].
- ◆ **-o** Specify the message log file. For more information, see “-o option” [*MobiLink Synchronization Reference*, page 13].
- ◆ **-v+** Verbose operation. For more information, see “-v option” [*MobiLink Synchronization Reference*, page 21].

Once you have executed the MobiLink synchronization client command, the output below appears to indicate that synchronization has succeeded. After synchronization, the remote database is populated with the data from the consolidated database.



Further reading

☞ For more information about dbmsync, see “MobiLink synchronization client” [*MobiLink Synchronization Reference*, page 36].

For more information about synchronization, see [“The synchronization process”](#) on page 21.

Lesson 5: Monitoring your MobiLink synchronization using log files

Once the tables have synchronized, you can view the progress of the synchronization using the two message log files you created with each command line, namely, *mlserver.mls* and *dbmlsync.out*.

❖ To find errors in a MobiLink synchronization log file

1. Open your log file in a text editor. For this tutorial, the log file is *mlserver.mls*.
2. Search the file for the string `Synchronization Server started`.
3. Scan down the left side of the file. A line beginning with *I.* contains an informational message, and a line beginning with *E.* contains an error message. For example:

```
I. 04/27 16:01:00. <Main>: ODBC isolation set to: Read Committed
I. 04/27 16:01:00. <Main>: Connected
I. 04/27 16:01:00. <Main>: Translated SQL:
                        SELECT COUNT(*) FROM ml_scripts_modified
E. 04/27 16:01:01. <Main>: Error: Unable to initialize communications stream 1: tcpip
I. 04/27 16:01:01. <Main>: Synchronization Server shutting down
I. 04/27 16:01:03. <Main>: Disconnected
I. 04/27 16:01:03. <Main>: Synchronization Server finished
```

4. Note that beside the *E.* in this example, there is the following text:

```
04/27 16:01:01. <Main>: Error: Unable to initialize
communications stream 1: tcpip.
```

This message indicates an error prior to the upload and download. There may be errors in the synchronization subscription or publication definitions.

5. Look for the clause that begins as follows:

```
Synchronization request from:
```

This clause indicates that a synchronization request has been established.

6. Look for the clause that begins `Working on a request`. This indicates that the client and server are communicating. You may get this message if you have specified a high level of verbosity.

❖ **To detect errors in your MobiLink synchronization client log file**

1. Open your log file in a text editor. For this tutorial, the log file is *dbmlsync.out*.
2. Search the file for the string `COMMIT`. If it appears, your synchronization was successful.
3. Search the file for the string `ROLLBACK`. If the transaction was rolled back, there were errors that prevented it from completing.
4. Scan down the left side of the file. If you see an *E.*, you have an error. If you don't have any errors, your synchronization has completed successfully.

Further reading

☞ For more information about MobiLink synchronization server log files, see [“Logging MobiLink synchronization server actions” on page 17](#).

Tutorial cleanup

You should remove tutorial materials from your computer.

❖ To remove tutorial materials from your computer

1. Close the Adaptive Server Anywhere, MobiLink, and synchronization client windows by right-clicking on each taskbar item and choosing Close.
2. Delete all tutorial-related data sources.
 - ◆ Choose Start ► Programs ► Sybase SQL Anywhere 9 ► Adaptive Server Anywhere ► ODBC Administrator.
The ODBC Data Source Administrator appears.
 - ◆ Select test_remote and test_consol from the list of User Data Sources. Click Remove.
 - ◆ Click OK to close the ODBC Data Source Administrator.
3. Delete the consolidated and remote databases.
 - ◆ Open Windows Explorer and navigate to the *Samples\MobiLink\AutoScripting* subdirectory of your SQL Anywhere 9 installation.
 - ◆ Delete *remote.db*, *remote.log*, *consol.db*, and *consol.log*.

Further reading

The following documentation sections are a good starting point for further reading:

☞ For more information about running the MobiLink synchronization server, see [“The MobiLink synchronization server” on page 16](#).

☞ For more information about synchronization scripting, see [“Writing Synchronization Scripts” on page 37](#) and [“Synchronization Events”](#) [*MobiLink Synchronization Reference*, page 83].

☞ For an introduction to other methods of synchronization such as timestamp, see [“Synchronization Techniques” on page 69](#).

☞ For information about testing your scripts in Sybase Central, see [“Testing script syntax” on page 64](#).

CHAPTER 19

Tutorial: Using MobiLink with an Oracle 8i Consolidated Database

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 an Adaptive Server Anywhere database.

This tutorial will guide you through the process of creating and synchronizing the two databases.

Contents

Topic:	page
Introduction	402
Lesson 1: Create your databases	403
Lesson 2: Starting the MobiLink synchronization server	409
Lesson 3: Running the MobiLink synchronization client	410
Summary	411
Further reading	412

Introduction

In this tutorial, you prepare an Oracle consolidated database and an Adaptive Server Anywhere remote database. You then synchronize the two databases using MobiLink.

Timing

The tutorial takes about 120 minutes.

Required software

- ◆ A full Adaptive Server Anywhere installation including Sybase Central.
- ◆ A full installation of MobiLink synchronization server and client.
- ◆ A full installation of Oracle Enterprise Edition 8i.
- ◆ The iAnywhere Solutions - Oracle 8, 8i and 9i driver.

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 synchronization 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.

Suggested background reading

☞ For more information about writing SQL scripts, see [“Tutorial: Writing SQL Scripts Using Sybase Central” on page 383](#).

☞ For more information about running the MobiLink synchronization server, see [“Synchronization Basics” on page 7](#).

Lesson 1: Create your databases

MobiLink synchronization requires that you have data a relational database, an ODBC data source for each database and two compatible databases.

SQL files

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

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 will appear in the SQL Plus output. 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, 'empl' );
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;
```

Copy the following code into SQL Plus and execute it. These SQL statements drop and create tables in the remote databases. If there are no tables to drop, an error will appear in the SQL Plus output. This will not affect processing. Synchronization subscriptions and publications are also inserted to define the synchronization parameters for the MobiLink synchronization server.

```

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';

```

ODBC data sources

You can now create ODBC data sources through which you connect to the Oracle consolidated database and the Adaptive Server Anywhere remote database. MobiLink requires ODBC data source to perform data synchronization.

The consolidated data source

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. You will set up the ODBC connections for the Adaptive Server Anywhere remote database later.

❖ To set up an ODBC data source for Oracle

1. Choose Start ► Programs ► Sybase SQL Anywhere 9 ► Adaptive Server 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 - Oracle 8, 8i and 9i Driver, 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 value 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. Enter a Default User Name. For this tutorial you can use **system**, or any User Name with sufficient rights to create objects. Click OK.
7. Click OK to close the ODBC Data Source Administrator.

MobiLink system tables

MobiLink comes with a script called *syncora.sql*, located in the *MobiLink\setup* subdirectory of your SQL Anywhere installation. *Syncora.sql* contains SQL statements, written in Oracle SQL, to prepare Oracle databases for use as consolidated databases. It creates a series of system tables, triggers, and procedures for use by MobiLink. The system tables are prefaced with *ML_*. MobiLink works with these tables during the synchronization process.

❖ Create MobiLink system tables within Oracle

1. Start SQL Plus. Choose Start ► Programs ► Oracle - OraHome81 ► Application Development ► SQL Plus.

Connect to your Oracle database by 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 9 installation. If there are spaces in your path, you should enclose the path and filename in quotation marks.

❖ To verify that the system tables are installed

1. Start SQL Plus. Choose Start ► Programs ► Oracle - OraHome81 ► 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 all of the objects shown in the following table are included, you can proceed to the next step.

OBJECT_NAME

ML_ADD_CONNECTION_SCRIPT
ML_ADD_DNET_CONNECTION_SCRIPT
ML_ADD_DNET_TABLE_SCRIPT
ML_ADD_JAVA_CONNECTION_SCRIPT
ML_ADD_JAVA_TABLE_SCRIPT
ML_ADD_LANG_CONNECTION_SCRIPT
ML_ADD_LANG_TABLE_SCRIPT
ML_ADD_TABLE_SCRIPT
ML_ADD_USER
ML_CONNECTION_SCRIPT
ML_CONNECTION_SCRIPT_TRIGGER
ML_SCRIPT
ML_SCRIPTS_MODIFIED
ML_SCRIPT_TRIGGER
ML_SCRIPT_VERSION
ML_SUBSCRIPTION
ML_TABLE
ML_TABLE_SCRIPT
ML_TABLE_SCRIPT_TRIGGER
ML_USER

Note

If any of the objects are missing, 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. However, do not drop system tables if there are any tables starting with ML_ other than the ones listed above.

❖ To drop the MobiLink system tables

1. Generate and run the drop statements:

◆ Run the following SQL statement in SQL Plus:

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

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

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.

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

```
@c:\OracleTut\drop.sql;
```

Replace *c:* 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 for Creating MobiLink system tables in Oracle.

The remote data source

❖ To initialize your remote database

1. Open a command prompt and navigate to your *OracleTut* directory, for example *c:\OracleTut*. Run the following command line:

```
dbinit remote.db
```

2. Verify the successful creation of the database by getting a listing of the contents of this directory. The file *remote.db* should appear in the directory listing.

❖ To create an ODBC data source for the remote database

1. Open a command prompt and navigate to your *OracleTut* directory. Run the following command line:

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

Replace *c:* with the location of your *OracleTut* directory.

❖ To verify your new data source

1. Choose Start ► Programs ► Sybase SQL Anywhere 9 ► Adaptive Server Anywhere ► ODBC Administrator.
The ODBC Data Source Administrator appears.
2. Click the User DSN tab.
3. Select *test_remote* from the list of data sources and click Configure.
4. Test *test_remote* by clicking the Test Connection button.
5. Click OK to close the ODBC Data Source Administrator.

Databases

In this procedure, you build a consolidated database using the *dbisql* command line utility. The *dbisql* utility helps you to execute SQL commands within your database. This procedure executes SQL statements within each database.

☞ For more information about *dbisql*, see “The Interactive SQL utility” [ASA Database Administration Guide, page 492].

❖ To create and populate tables in the consolidated database

1. Start SQL Plus and connect to your consolidated database. Choose Start ► Programs ► Oracle - OraHome81 ► Application Development ► SQL Plus.
2. Execute the SQL statements in *build_consol.sql* by running the following command:

```
@c:\OracleTut\build_consol.sql;
```

Replace *c:* with the location of your *OracleTut* directory. If the path contains spaces, enclose the path and filename in double quotes.

3. Verify the successful creation of each of the tables through SQL Plus directly from within the application. Run the following SQL statements:

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

4. Leave the consolidated database running.

❖ To create tables and synchronization information in the remote database

1. Open a command prompt and navigate to your *OracleTut* directory. Run the following command line:

```
dbisql -c "dsn=test_remote" build_remote.sql
```

The *dbisql* plug-in starts the remote database and executes the SQL statements in *build_remote.sql*.

2. Verify the successful creation of the emp and cust tables using Interactive SQL or Sybase Central.
3. Leave the consolidated and remote databases running.

Lesson 2: Starting the MobiLink synchronization server

☞ The MobiLink synchronization server can now be started from a command prompt. Since MobiLink synchronization server is a client to the consolidated database, your consolidated database must be started prior to starting MobiLink. If you shut down your consolidated database following Lesson 1, you should restart the database.

❖ To start the MobiLink synchronization server

1. Ensure that your consolidated database is running.
2. Open a command prompt and navigate to your *OracleTut* directory. Run the following command line:

```
dbmlsrv9 -c "dsn=ora_consol;pwd=manager" -o mlserver.mls -v+
        -za -zu+
```

This command line specifies the following options:

- ◆ **-c** Specifies connection parameters. Note that we only use the password as the User ID is contained in the DSN. For more information, see “-c option” [*MobiLink Synchronization Reference*, page 10].
- ◆ **-o** Specifies the message log file. For more information, see “-o option” [*MobiLink Synchronization Reference*, page 13].
- ◆ **-v+** Sets verbose logging on. For more information, see “-v option” [*MobiLink Synchronization Reference*, page 21].
- ◆ **-dl** Sets the display log feature ON.
- ◆ **-za** Turns automated scripting ON. For more information, see “-za option” [*MobiLink Synchronization Reference*, page 28].
- ◆ **-zu+** Automates the user authentication process. For more information, see “-zu option” [*MobiLink Synchronization Reference*, page 31].

Further reading

☞ For more information about dbmlsrv9, see “[The MobiLink synchronization server](#)” on page 16 and “MobiLink synchronization server” [*MobiLink Synchronization Reference*, page 4].

Lesson 3: Running 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 on the *dbmlsync* command line using the *-c* option. These parameters are for the *remote* database.

❖ To start the MobiLink client

1. Ensure that the MobiLink synchronization server is started.
2. Open a command prompt and navigate to your *OracleTut* directory. Run the following command line:

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

This command line specifies the following options:

- ◆ **-c** Supply database connection parameters. For more information, see “-c option” [*MobiLink Synchronization Reference*, page 10].
- ◆ **-o** Specify the message log file. For more information, see “-o option” [*MobiLink Synchronization Reference*, page 13].
- ◆ **-v+** Verbose operation. For more information, see “-v option” [*MobiLink Synchronization Reference*, page 21].
- ◆ **-e** Extended options. Specifying “SendColumnNames=ON” sends column names to MobiLink. For more information, see “-e option” [*MobiLink Synchronization Reference*, page 12].

Further reading

☞ For more information about *dbmlsync*, see “MobiLink synchronization client” [*MobiLink Synchronization Reference*, page 36].

Summary

During this tutorial, you accomplished the following tasks.

- ◆ Created a new Adaptive Server Anywhere database to serve as a remote database.
- ◆ Started a MobiLink synchronization server to work with your consolidated Oracle database.
- ◆ Started the MobiLink synchronization client and synchronized the remote database with the consolidated Oracle database.

Learning
accomplishments

In this tutorial, you gained:

- ◆ Familiarity with the MobiLink synchronization server and client and how they work with an Oracle database.
- ◆ Competence in executing MobiLink server and client commands.

Further reading

The following documentation areas are good starting points for further reading:

☞ For more information about running the MobiLink synchronization server, see [“Running the MobiLink synchronization server” on page 16](#).

☞ For more information about synchronization scripting, see [“Writing Synchronization Scripts” on page 37](#), and “Synchronization Events” [*MobiLink Synchronization Reference*, page 83].

☞ For an introduction to other methods of synchronization such as timestamp, see [“Synchronization Techniques” on page 69](#).

CHAPTER 20

The Contact Sample Application

About this chapter

This chapter uses the Contact MobiLink sample application 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.

Contents

Topic:	page
Introduction	414
Setup	415
Tables in the Contact databases	417
Users in the Contact sample	420
Synchronization	421
Monitoring statistics and errors in the Contact sample	428

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 an Adaptive Server Anywhere consolidated database and two Adaptive Server Anywhere remote databases. It illustrates several common synchronization techniques. To get the most out of this example, you should study the sample application as you read.

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

The Contact sample is in the *Samples\MobiLink>Contact* subdirectory of your SQL Anywhere installation. For an overview, see *Samples\MobiLink>Contact\readme.txt*.

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 are shared with the remote databases.
- ◆ **Row subsets** All of the columns but only one of the rows of the SalesRep table are shared with each remote database.
 For more information, see [“Partitioning rows among remote databases” on page 77](#).
- ◆ **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 synchronization” on page 72](#).

Setup

A batch file named *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. Open a command prompt and navigate to the *Samples\MobiLink>Contact* subdirectory of your SQL Anywhere installation.
2. Run *build.bat* (Windows) or *build.sh* (Unix).

Running the Contact sample

The Contact sample includes batch files that carry out initial synchronizations and illustrate MobiLink synchronization server and dbmlsync command lines. You can examine the contents of the following batch files, located in the *Samples\MobiLink>Contact* subdirectory of your SQL Anywhere 9 installation, in a text editor:

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

❖ To run the Contact sample

1. Start the MobiLink synchronization server.

- ◆ Open a command prompt and navigate to the *Samples\MobiLink>Contact* subdirectory of your SQL Anywhere 9 installation and execute the following command:

```
step1
```

This command runs a batch file that starts the MobiLink synchronization 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.

- ◆ Open a command prompt and navigate to the *Samples\MobiLink>Contact* subdirectory of your SQL Anywhere 9 installation and execute the following command:

```
step2
```

This is a batch file that synchronizes both remote databases.

3. Shut down the MobiLink synchronization server.

- ◆ Open a command prompt and navigate to the *Samples\MobiLink>Contact* subdirectory of your SQL Anywhere 9 installation and execute the following command:

```
step3
```

This is a batch file that shuts down the MobiLink synchronization 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:

- ◆ *Samples\MobiLink\Contact\build_consol.sql*
- ◆ *Samples\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 SalesRep occupies one row in this 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 a `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 referencing 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 customer.
- ◆ **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 holding an identifying number for the product.
- ◆ **name** The name of the individual 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 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.

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\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 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 dbmluser utility.
- ◆ During synchronization, the MobiLink password for the connecting user is supplied on the dbmlsync command line listed in *Samples\MobiLink\Contact\step2.bat*.
- ◆ The MobiLink synchronization 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” on page 74](#).

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 synchronization” on page 72](#).

☞ For more information, see [“Partitioning rows among remote databases” on page 77](#).

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 downloads only customers assigned to a particular 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:

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

Conflict detection is not carried out on this table.

- ◆ **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 will be 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** 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 explicitly 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 Sam Singer takes an order for 20 baseball hats (product ID 400), he will change the quantity from 90 to 70. If Pam 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 Sam Singer synchronizes his changes, the quantity column in the consolidated database is changed from 90 to 70. When Pam 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 *Samples\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 21

The CustDB Sample Application

About this chapter

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

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

Contents

Topic:	page
Introduction	430
Setup	432
Tables in the CustDB databases	440
Users in the CustDB sample	443
Synchronization	444
Maintaining the customer and order primary key pools	448
Further reading	450

Introduction

This chapter introduces you to the CustDB (Customer Database) MobiLink sample application. CustDB is a sales-status application.

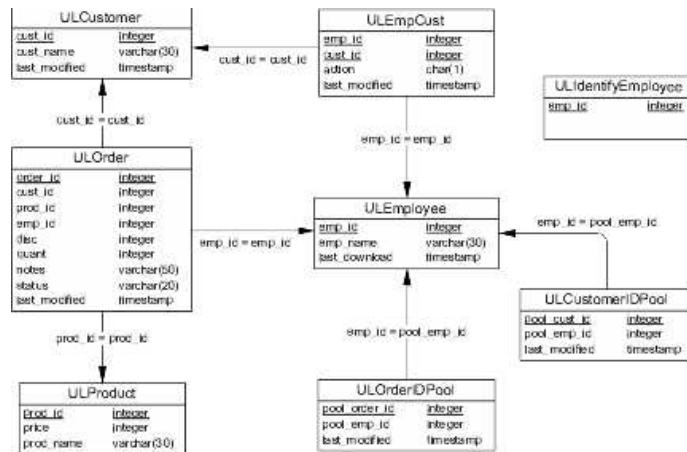
The CustDB sample is a valuable resource for the MobiLink developer. It provides you with examples of how to implement many of the techniques you will need to develop MobiLink applications.

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 432](#).

Following is the schema of CustDB:



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.
- ◆ 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” on page 77](#).
- ◆ **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 synchronization” on page 72](#).
- ◆ **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” on page 74](#).
- ◆ **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 [“Maintaining unique primary keys using key pools” on page 86](#).
 - ☞ For other ways to ensure that primary keys are unique, see [“Maintaining unique primary keys” on page 81](#).

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\MobiLink\CustDB* subdirectory of your SQL Anywhere installation.
- ◆ The application code, located in *Samples\UltraLite\CustDB*.
- ◆ Platform-specific user interface code, located in subdirectories of *Samples\UltraLite\CustDB* named for each operating system.

Setting up the CustDB consolidated database

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

The following SQL scripts are provided in the *Samples\MobiLink\CustDB* subdirectory of your SQL Anywhere 9 installation to build the consolidated database on any of these platforms:

- ◆ For an Adaptive Server Anywhere database, the file is *custdb.sql*.
- ◆ For an IBM DB2 database, the file is *custdb2.sql*.
- ◆ For an Adaptive Server Enterprise database, the file is *custase.sql*.
- ◆ For a Microsoft SQL Server database, the file is *custmss.sql*.
- ◆ For an Oracle database, the file is *custora.sql*.

Creating a consolidated database

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

For databases other than Adaptive Server Anywhere databases, you will first need to run a script to add the MobiLink system tables. This platform-specific script is located in the *MobiLink\setup* subdirectory of your SQL Anywhere 9 installation.

☞ For more information about preparing a database for use as a consolidated database, see [“Setting up a consolidated database” on page 11](#).

❖ **To set up a consolidated database (Adaptive Server Enterprise, Oracle or SQL Server)**

1. Create the consolidated database.
2. Add the MobiLink system tables by running one of the following SQL scripts, located in the *MobiLink\setup* subdirectory of your SQL Anywhere 9 installation:
 - ◆ For an Adaptive Server Enterprise consolidated database prior to version 12.5, run *syncase.sql*. Otherwise, run *syncase125.sql*.
 - ◆ For an Oracle consolidated database, run *syncora.sql*.
 - ◆ For a SQL Server consolidated database, run *syncmss.sql*.
3. Add tables to the CustDB database by running one of the following SQL scripts, located in the *Samples\MobiLink\CustDB* subdirectory of your SQL Anywhere 9 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 ► Sybase SQL Anywhere 9 ► Adaptive Server 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.
The default values are DBA and SQL.
 - ◆ Click the Database tab. Browse to the location of your database file.

❖ To set up a consolidated database (Adaptive Server Anywhere)

1. Create the consolidated database:

Navigate to the *Samples\MobiLink\CustDB* subdirectory of your SQL Anywhere 9 installation and run the following command line:

```
dbinit consol.db
```

2. Add tables to the CustDB database by running *custdb.sql*, located in the *Samples\MobiLink\CustDB* subdirectory of your SQL Anywhere 9 installation.

- ◆ Choose Start ► Programs ► Sybase SQL Anywhere 9 ► Sybase Central.
- ◆ In the right pane of Sybase Central, right-click Adaptive Server Anywhere 9 and connect to the consolidated database you have created. The default user ID and password are DBA and SQL.
- ◆ In the right pane, right-click the consolidated database and select Open Interactive SQL from the popup menu.
- ◆ In Interactive SQL, select File ► Run Script. Browse to *custdb.sql*. Click Open.
- ◆ Close Interactive SQL.

3. Create an ODBC data source called CustDB that references your database on the client machine.

- ◆ In Sybase Central, select Tools ► Adaptive Server Anywhere 9 ► Open ODBC Administrator.
- ◆ Click Add.
- ◆ Select Adaptive Server Anywhere 9.0 Driver. Click Finish.
- ◆ Name the ODBC data source CustDB.
- ◆ Click the Login tab. Enter the user ID and password for your database. The default values are DBA and SQL.
- ◆ Click the Database tab. Browse to the location of your database file.

❖ To set up a consolidated database (IBM DB2)

1. Create a DB2 database on the DB2 server. Ensure that the default table space (usually called USERSPACE1) uses 9 Kb pages.

If the default table space does not use 9 Kb pages, complete the following steps:

- ◆ Delete the default table space, USERSPACE1.
- ◆ Verify that at least one of your buffer pools has 9 Kb pages. If not, create a buffer pool with 9 Kb pages and restart the database to activate it.
- ◆ Create a new table space with 9 Kb pages and name it USERSPACE1. For more information, consult your DB2 documentation.

2. Add the MobiLink system tables using the file *MobiLink\setup\syncdb2long.sql*. (If you are using a version of DB2 prior to 6.5, use *syncdb2.sql*.)

- ◆ Change the connect command in *syncdb2long.sql*. Replace *DB2Database* with the name of your ODBC data source. In this example, the ODBC data source is CustDB. You could also add the user name and password as follows. Replace *userid* and *password* with your user name and password.

```
connect to DB2Database 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
```

3. Copy *custdb2.class* to the *SQLLIB\FUNCTION* directory on your DB2 server machine.

4. Add 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
```

-
5. Create an ODBC data source called CustDB that references the DB2 database on the DB2 client machine.
 - ◆ Choose Start ► Programs ► Sybase SQL Anywhere 9 ► Adaptive Server Anywhere ► ODBC Administrator.
 - ◆ Click Add.
 - ◆ Select IBM DB2 ODBC Driver. Click Finish.
 - ◆ Name the ODBC data source CustDB.
 - ◆ Click the Login tab. Enter the user ID and password for your database. The default values are DBA and SQL.
 - ◆ Click the Database tab. Browse to the location of your database file.
 6. Run the *custdb2setuplong* Java application on the DB2 client machine as follows. If you are using a version of DB2 prior to 6.5, use *custdb2setup*. 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
```

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 the *Samples\UltraLite\CustDB* subdirectory of your SQL Anywhere 9 installation. 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 *custdbapi.cpp* contains the C++ API logic.

- ◆ **User-interface features** These features are stored separately, in platform-specific subdirectories of *Samples\UltraLite\CustDB*.

You will complete the following steps in order 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 synchronization 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 and navigate to the *Samples\MobiLink\CustDB* subdirectory of your SQL Anywhere 9 installation. Run the following command line, 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 synchronization server:
 - ◆ For a DB2 database, run the following command at a command prompt:

```
dbmlsrv9 -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 9 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 443](#).
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 synchronization server are running.
 - ◆ Place the Palm device in its cradle.
 - ◆ Press the HotSync button to synchronize.

Clean up

You may want to reset the data in the CustDB database in order to restart the sample. To revert the data in the CustDB UltraLite database to its original state, complete the following steps.

❖ **To reset the data in the sample application**

1. Install the ULUtil on your device:
 - ◆ For a Palm device, start Palm Desktop on your PC.
 - ◆ Click Install on the Palm Desktop toolbar.
 - ◆ Click Add. Browse to *ulutil.prc* in the *UltraLite\palm\68k* subdirectory of your SQL Anywhere 9 installation.
 - ◆ Click Done.
 - ◆ HotSync your Palm device.
2. Delete the data using ULUtil:
 - ◆ For a Palm device, tap the ULUtil icon.
 - ◆ Select CustDB and tap Delete Data.
 - ◆ HotSync your Palm device.

Tables in the CustDB databases

The table definitions for the CustDB database are in platform-specific files in the *Samples\MobiLink\CustDB* subdirectory of your SQL Anywhere 9 installation.

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 identifying 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 holds a unique integer identifying the product.
- ◆ **price** An integer identifying the unit price.
- ◆ **prod_name** A 30-character string containing 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_cust_id** An integer column containing the customer ID that will be used for a new customer 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 will be downloaded. If the employee needs a new customer's orders, inserting the employee ID and customer ID will force 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 exist only in the consolidated database. They are for conflict resolution and contain the same columns as ULOrder. In Adaptive Server 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” [ASA SQL Reference, page 37].

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 do the following:
 - 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',  
  '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:

- ◆ Only approved orders are downloaded.
- ◆ Orders can be modified at both the consolidated and remote databases.
- ◆ Each remote database contains only the orders assigned to an employee.

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

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

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

```
CALL ULOrderDownload( ?, ? )
```

The **ULOrderDownload** procedure for CustDB is as follows:

```

ALTER 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, ULEmpCust ec
WHERE o.cust_id = ec.cust_id
AND ( ( o.status = 'Approved' AND o.last_modified > ? )
OR ( ec.action = 'D' ) )
AND ec.emp_id = ?

```

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",
                        "disc", "quant", "notes", "status" )
VALUES ( ?, ?, ?, ?, ?, ?, ? )

```

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

```

UPDATE ULOrder SET cust_id=?, prod_id=?, emp_id=?, disc=?,
                  quant=?, notes=?, status=?
WHERE order_id = ?

```

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

```

DELETE FROM "ULOrder" WHERE "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 = ?

```

- ◆ **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( ?, ?, ?, ?, ?, ?, ?, ? )
```

- ◆ **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( ?, ?, ?, ?, ?, ?, ?, ? )
```

Conflict resolution

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

```
CALL ULResolveOrderConflict
```

The **ULResolveOrderConflict** procedure for CustDB is as follows:

```
ALTER 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 >
?
```

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 ( ?, ?
)
```

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

```
UPDATE ULCustomer SET cust_name = ?
WHERE "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 = ?
```

Synchronizing products in the CustDB sample

Business rules

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 > ?
AND pool_emp_id = ?
```

Uploads

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

```
INSERT INTO ULCustomerIDPool ( pool_cust_id ) VALUES( ? )
```

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

```
DELETE FROM ULCustomerIDPool WHERE pool_cust_id = ?
```

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

```
CALL ULCustomerIDPool_maintain( ? )
```

The **UL_CustomerIDPool_maintain** procedure for CustDB is as follows:

```
ALTER 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 > ?
AND pool_emp_id = ?
```

Uploads

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

```
CALL ULOrderIDPool_maintain( ? )
```

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( ? )
```

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

```
DELETE FROM ULOrderIDPool WHERE pool_order_id = ?
```

Further reading

The following documentation sections are good starting points for further reading:

- ☞ For more information about script types, see [“Script types” on page 46](#).
- ☞ For reference material, including detailed information about each script and its parameters, see “Synchronization Events” [*MobiLink Synchronization Reference*, page 83].

Index

Symbols

-MLAutoLoadPath option	
about	255
-MLDomConfigFile option	
about	255
-notifier	
dbmlsrv9 option	145
-sl dnet option	
user-defined start classes	263
using -MLAutoLoadPath	255
using -MLDomConfigFile	255
-sl java option	
user-defined start classes	237
-v option	
MobiLink [dbmlsync] performance	288
.NET	
about support in MobiLink	251
MobiLink API reference	269
MobiLink data types	261
synchronization logic	31
synchronization scripts for MobiLink	251
.NET MobiLink API	
API reference	269
benefits	34
.NET classes	
instantiation for .NET synchronization	
logic	260
.NET synchronization logic	
.NET class instantiations	260
about	31
API	269
DBCommand	269
DBConnection	271
DBConnectionContext	271
DBParameter	272
DBParameterCollection	273
DBRowReader	276
debugging	259
InOutInteger	282
LogCallback	277

LogMessage	277
MessageType	277
methods	262
sample	266
ServerContext	277
ServerException	279
setup	253
ShutdownCallback	279
SQLType	280
supported languages	252
#hook_dict table	
about	196
unique primary keys	84

A

ActiveSync	
CREATE SYNCHRONIZATION	
USER statement for MobiLink	
Adaptive Server Anywhere	
clients	190
deploying MobiLink UltraLite	
applications	225
installing the MobiLink provider for	
Adaptive Server Anywhere	
clients	191
installing the MobiLink provider for	
UltraLite clients	223
MobiLink Adaptive Server Anywhere	
clients	189
registering applications for Adaptive	
Server Anywhere clients	192
registering applications for UltraLite	
clients	224
Adaptive Server Anywhere	
as MobiLink clients	19
as MobiLink consolidated database	12
Adaptive Server Anywhere clients	
MobiLink	167
Adaptive Server Enterprise	
as MobiLink consolidated database	12
MobiLink synchronization	66
StaticCursorLongColBuffLen	66

- add connection script wizard
 - using 51
 - add service wizard
 - using 331
 - add synchronized table wizard
 - using 51
 - add synchronizing table script wizard
 - using 52, 389
 - add user wizard
 - using 110
 - add version wizard
 - using 50, 389
 - adding
 - articles 175
 - columns to remote MobiLink
 - databases 100
 - MobiLink users to a remote database 178
 - synchronization scripts with Sybase
 - Central 51
 - tables to remote MobiLink databases 100
 - adding a script version 50
 - adding and deleting scripts in your
 - consolidated database 51
 - adding MobiLink users to a remote database 178
 - adding synchronization scripts
 - using stored procedures 52
 - altering
 - articles 175
 - publications 175
 - synchronization subscriptions 183
 - altering MobiLink subscriptions 183
 - Apache
 - configuring servlet Redirector for MobiLink 325
 - Apache Tomcat
 - servlet Redirector 325
 - API reference
 - MobiLink .NET API 269
 - MobiLink Java API 246
 - applications
 - differentiating MobiLink scripts 49
 - article creation wizard
 - using 175
 - articles
 - adding 175
 - altering 175
 - creating 171
 - MobiLink synchronization
 - subscriptions 182
 - removing 175
 - assemblies
 - implementing in MobiLink 255
 - locating in MobiLink .NET
 - synchronization logic 253
 - authenticate_user
 - about 114
 - automatic synchronization script
 - generation 40
 - automating scripts
 - MobiLink synchronization 40
- B**
- begin_connection
 - example 444
 - blob cache size
 - MobiLink performance 287
 - BLOBs
 - downloaded from Adaptive Server Enterprise 66
 - bottlenecks
 - MobiLink performance 290
- C**
- C++
 - support in MobiLink .NET 252
 - C#
 - support in MobiLink .NET 252
 - cascading deletes
 - during MobiLink synchronization 28
 - Certicom
 - obtaining certificates 360
 - certificate authorities 351
 - certificate chains 352
 - certificates
 - sample certificates for MobiLink 346
 - chains of certificates
 - using 343
 - CHAR data type
 - MobiLink and other DBMSs 67
 - ciphers
 - MobiLink transport-layer security 338

CLASSPATH environment variable	for use with the Notifier	143
MobiLink Java synchronization logic	conflict detection	
229	MobiLink	90
clients	MobiLink statement-based uploads	90
Adaptive Server Anywhere as	conflict resolution	
MobiLink	Contact sample	425
Adaptive Server Anywhere MobiLink	CustDB sample	447
clients	forcing in MobiLink	92
MobiLink synchronization	MobiLink	90
UltraLite applications as MobiLink	MobiLink conflict detection	90
UltraLite MobiLink clients	MobiLink statement-based uploads	90
columns	user-specific logic	93
adding to remote MobiLink databases	ConflictRetries synchronization option	
100	about	187
commit_state column	conflicts	
about	MobiLink	90
communications	connection parameters	
specifying for MobiLink	priority order	180
communications faults	connection scripts	
MobiLink synchronization recovery	about	46
concurrency	adding with Sybase Central	51
MobiLink performance	defined	46
MobiLink synchronization	consolidated databases	
MobiLink upload-stream processing	Adaptive Server Anywhere as	
conduit	MobiLink	12
dbcond9.exe	Adaptive Server Enterprise as	
deploying	MobiLink	12
deploying UltraLite applications	adding synchronization scripts to	51
HotSync synchronization	compatibility issues for MobiLink	
installing	applications	65
testing	creating MobiLink	11
conduit installation utility	databases other than Adaptive Server	
about	Anywhere	65
config.notifier	DBMS dependencies	65
about	IBM DB2 as MobiLink	13
configuring	MobiLink	10
Microsoft web servers	MobiLink user names	20
Netscape web servers	Oracle as MobiLink	13
Redirectors (all versions)	relating tables to MobiLink remote	
server-initiated synchronization	tables	11
servlet Redirector	server-initiated synchronization	143
Tomcat	SQL Server as MobiLink	14
configuring Adaptive Server Anywhere	supportedr	10
remote databases for	constructors	
ActiveSync	MobiLink synchronization	233, 261
configuring MobiLink user properties	Contact MobiLink sample	
configuring the consolidated database	about	414

building	415	cursor scripts	
Contact table	423	defined	46
Customer table	421	custase.sql	
monitoring statistics	428	location	432
Product table	425	CustDB application	
running	415	MobiLink sample application	429
SalesRep table	421	synchronization scripts	432
tables	417	CustDB database	
users	420	DB2	432
contd_timeout stream parameter		MobiLink sample application	429
synchronizing across firewalls	316	CustDB MobiLink sample	
contention		tables	440
MobiLink performance	286	ULCustomer table	446
MobiLink performance explanation		ULOrder table	444
291		ULProduct table	447
conventions		users	443
documentation	xii	custdb.sqc	
create database wizard		location	436
using	385	custdb.sql	
CREATE SYNCHRONIZATION		location	432
SUBSCRIPTION statement		custmss.sql	
ActiveSync for MobiLink Adaptive		location	432
Server Anywhere clients	190	customizing	
CREATE SYNCHRONIZATION USER		MobiLink	194
statement		customizing a prototype remote database	
ActiveSync for MobiLink Adaptive		168	
Server Anywhere clients	190	custora.sql	
creating		location	432
Adaptive Server Anywhere remote			
databases	168	D	
articles	171	daemon	
MobiLink consolidated databases	11	running MobiLink as a	329
MobiLink users in remote databases		data entry	
178		synchronization techniques	94
publications	171	data movement technologies	
publications with column-wise		MobiLink synchronization	7
partitioning	172	data sources	
publications with row-wise		ODBC for MobiLink synchronization	
partitioning	173	12	
publications with whole tables	171	data types	
creating a consolidated database	11	MobiLink .NET and SQL	261
creating a remote database		MobiLink Java and SQL	233
Adaptive Server Anywhere clients	168	database connections	
creating MobiLink users	178	MobiLink performance	293
creating the certificates	354	database schemas	
cryptography		relating consolidated tables to	
public key	338	MobiLink remote tables	11

-
- databases
 - MobiLink consolidated 10
 - MobiLink synchronization
 - requirements for consolidated 10
 - synchronizing with MobiLink 7, 337
 - DB2
 - as MobiLink consolidated database 13
 - consolidated database 432
 - CustDB database 432
 - session-wide variables 66
 - dbasinst utility
 - installing the MobiLink provider for
 - ActiveSync for Adaptive Server Anywhere clients 191
 - installing the MobiLink provider for
 - ActiveSync for UltraLite clients 223
 - DBCommand
 - MobiLink .NET API 269
 - dbcond9 utility
 - deploying 212
 - HotSync conduit 212
 - DBConnection
 - MobiLink .NET API 271
 - DBConnectionContext
 - MobiLink .NET API 271
 - MobiLink Java API 246
 - dbhsync9.dll
 - HotSync conduit 211
 - dblgen9.dll
 - HotSync conduit deployment 211
 - dblsn
 - Listener utility for Windows 154
 - dblsn.txt
 - MobiLink Listener default parameters 160
 - dbmlhttp9.dll
 - deploying UltraLite applications 211
 - dbmlhttps9.dll
 - deploying UltraLite applications 211
 - dbmlmon.exe
 - monitoring MobiLink 297
 - dbmlsock9.dll
 - deploying UltraLite applications 211
 - dbmlsrv9
 - notifier option 145
 - automating script generation 40
 - using 16
 - dbmlstop utility
 - MobiLink 17
 - using 16
 - dbmlsync utility
 - d option 187
 - ActiveSync for MobiLink Adaptive Server Anywhere clients 189
 - changing passwords 113
 - concurrency 187
 - customizing MobiLink synchronization 194
 - example 185
 - multiple users 185
 - passwords 112
 - permissions 185
 - transaction logs 187
 - using 185
 - using version 7 clients 200
 - writing your own 188
 - dbmltts9.dll
 - deploying UltraLite applications 211
 - dbmluser utility
 - using 112
 - DBMS-dependent scripts 65
 - DBParameter
 - MobiLink .NET API 272
 - DBParameterCollection
 - MobiLink .NET API 273
 - DBRowReader
 - MobiLink .NET API 276
 - dbser9.dll
 - deploying UltraLite applications 211
 - dbtools.h
 - dbmlsync features 188
 - synchronization 188
 - DDL statements
 - remote MobiLink databases 100
 - deadlocks
 - MobiLink upload-stream processing 27
 - debugging
 - .NET synchronization logic 259
 - MobiLink connections 336
 - MobiLink synchronization server log 17
 - MobiLink synchronization using Java

classes	235	Java user-defined start classes	237
DECIMAL data type		documentation	
MobiLink and Adaptive Server Enterprise	67	conventions	xii
default global autoincrement		SQL Anywhere Studio	x
declaring	83	domain configuration files	
deletes		about	256
stopping upload of using MobiLink		download acknowledgement	
193		MobiLink performance	287
deleting		download cache size	
all rows in a remote MobiLink table	58	MobiLink performance	287
articles	175	download stream	
publications	177	defined	21
deleting rows		events	56
synchronization	58	failed downloads	96
synchronization techniques	95	MobiLink performance	289
deleting rows with the		MobiLink transactions	25
download_delete_cursor script		download-only synchronization	
58		about	30
deploying		download_cursor	
applications that use ActiveSync for		Contact sample	424, 425
UltraLite clients	225	CustDB sample	447
MobiLink ASA remote databases	168	disjoint partitioning	77
MobiLink remote database sample	168	example	444
MobiLink synchronization conduit for		example using a stored procedure call	
Palm	216	97	
troubleshooting MobiLink deployment		partitioning child tables	80
170		partitioning with overlaps	78
UltraLite Palm applications	216	performance	289
deployment options		timestamp-based synchronization	73
MobiLink remote databases	19	using a stored procedure call	97
deprecated features		download_cursor table script	
MobiLink differences from version 7		Contact sample	422
200		CustDB sample	446
development tips		download_delete_cursor	
synchronization	71	about	58
dial-up networking		Contact sample	422, 424, 425
about	216	CustDB sample	447
configuring	219	disjoint partitioning	77
digital certificates	341	example using a stored procedure call	
direct inserts of scripts	53	97	
disjoint partitioning		partitioning child tables	80
defined	77	partitioning with overlaps	78
synchronization	77	performance	289
distributed databases		using a stored procedure call	97
MobiLink synchronization	7	download_delete_cursor	
DMLStartClasses		timestamp-based	
		synchronization	72

downloading a result set from a stored procedure call		MobiLink	22
synchronization techniques	97	synchronization logic and	38
downloading data		events during download	56
file-based downloads in MobiLink	117	events during upload	54
downloading rows		example scripts	
synchronization scripts	56	generating	41
downloads		example scripts for UltraLite	43
file-based MobiLink	117	example synchronization script	
DROP PUBLICATION statement		generation	41
about	177	example_download_cursor	
DROP SYNCHRONIZATION		about	43
SUBSCRIPTION statement		example_upload_cursor	
about	183	about	43
dropping		examples	
MobiLink subscriptions	183	synchronization scripts	43
MobiLink users from a remote		extended options	
database	180	configuring at remote databases	179
dropping publications	177	priority order	180
		extended options for performance tuning	
		MobiLink	186
E		F	
encryption		failed downloads	
HotSync synchronization	214	synchronization techniques	96
MobiLink	338	failover	
enterprise root certificates	353	Redirector	314
creating	354	faults	
error handling		MobiLink synchronization recovery	26
during MobiLink synchronization	62	feedback	
errors		documentation	xvi
handling during MobiLink		providing	xvi
synchronization	62	file-based downloads	
multiple	63	about	117
recording	62	file-definition database	
event hooks		defined	119
#hook_dict table	196	firewalls	
connections	196	configuring MobiLink clients	316
event arguments	196	configuring MobiLink synchronization	
fatal errors	196	server	316
ignoring errors	197	routing requests	314
MobiLink	194	forced conflict resolution	
procedure owner	196	MobiLink	92
using	195	MobiLink statement-based uploads	92
event names		forcing conflicts	
defined	38	MobiLink	92
events		MobiLink statement-based uploads	92
Adaptive Server Anywhere client	194	fundamental rules	
introduction to MobiLink events	38		

- MobiLink 71
- G**
- generating example scripts 41
- generating scripts automatically 40
- getServerContext method
 - DBConnectionContext class 246, 272
- global assembly cache
 - implementing in MobiLink 255
- global autoincrement
 - algorithm 85
 - declaring 83
 - setting GLOBAL_DATABASE_ID 83
 - using to generate unique values 82
- GLOBAL_DATABASE_ID option
 - setting in MobiLink 83
- globally signed certificates 358
- H**
- handle_error
 - synchronization scripts 62
- handling deletes
 - synchronization techniques 95
- handling failed downloads
 - synchronization techniques 96
- handling multiple errors on a single SQL statement 63
- hooks
 - Adaptive Server Anywhere client 194
- host stream parameter
 - synchronizing across firewalls 316
- HotSync conduit
 - configuring 214
 - testing 213
- HotSync synchronization
 - about 209
 - architecture 210
 - Palm Computing Platform 211
- how remote tables relate to consolidated tables 11
- HTTP synchronization
 - Palm Computing Platform 217
- HTTPS synchronization
 - Palm Computing Platform 217
- I**
- iaredirect.dll
 - configuring the ISAPI Redirector 323
 - configuring the NSAPI Redirector 320
- IBM DB2
 - as MobiLink consolidated database 13
 - session-wide variables 66
- icons
 - used in manuals xiv
- IIS
 - configuring for ISAPI 323
- indexes
 - MobiLink performance 289
- initiating
 - MobiLink synchronization from
 - UltraLite applications 19
 - synchronization 185
- initiating synchronization from an application 188
- InOutByteArray
 - MobiLink Java API 246
- InOutInteger
 - MobiLink Java API 247
- InOutString
 - MobiLink Java API 247
- inserting
 - scripts in MobiLink 53
- installing
 - MobiLink provider for ActiveSync for
 - Adaptive Server Anywhere
 - clients 191
 - MobiLink provider for ActiveSync for
 - UltraLite clients 223
 - servlets into EAServer 325
 - introduction to synchronization scripts 38
 - invoking transport-layer security 346
- iPlanet
 - configuring for the NSAPI Redirector 320
- ISAPI Redirector
 - calling 323
 - configuring 323
- isolation levels
 - MobiLink default 21
- J**
- Java
 - MobiLink data types 233
 - MobiLink Java API reference 246

-
- synchronization logic 31
 - synchronization scripts for MobiLink 227
 - Java classes
 - instantiation for Java synchronization logic 232
 - Java MobiLink API
 - benefits 33
 - Java synchronization logic
 - about 31
 - API 246
 - DBConnectionContext 246
 - InOutByteArray 246
 - InOutInteger 247, 250
 - InOutString 247
 - Java class instantiations 232
 - LogListener 247
 - LogMessage 247
 - methods 234
 - sample 240
 - ServerContext 248
 - ServerException 250
 - setup 229
 - ShutdownListener 250
 - specifying in MobiLink server
 - command line 231
 - Java vs. SQL synchronization logic
 - MobiLink performance 288
 - Javadoc
 - MobiLink 246
 - K**
 - key pools
 - MobiLink synchronization application 86
 - L**
 - last download timestamp
 - about 72
 - Contact sample 423
 - maintaining 423
 - script parameter 48
 - last modified column
 - about 72
 - last_download_timestamp
 - script parameter 48
 - library functions
 - ULSynchronize 19
 - Listener utility
 - about 154
 - Listeners
 - about 138
 - configure and start 154
 - default parameters file 160
 - limitations of UDP Listeners 163
 - limitations on CE or PCs 163
 - Listener utility for Windows (dblsn) 154
 - Palm devices 160
 - SDK 162
 - Windows 154
 - loading assemblies in MobiLink 255
 - locking
 - MobiLink synchronization 187
 - LockTables synchronization option
 - about 187
 - log files
 - MobiLink 396
 - MobiLink synchronization server 17
 - synchronization 215
 - LogCallback
 - MobiLink .NET API 277
 - logging
 - MobiLink performance 288
 - MobiLink synchronization server
 - actions 17
 - LogListener
 - MobiLink Java API 247
 - LogMessage
 - MobiLink .NET API 277
 - MobiLink Java API 247
 - M**
 - maintaining unique primary keys using
 - global autoincrement 82
 - maintaining unique primary keys using
 - key pools 86
 - maintaining unique primary keys using
 - UUIDs 81
 - making a new self-signed certificate 348
 - many-to-many relationships
 - partitioning 78
 - synchronization 78
 - MessageType

MobiLink .NET API	277	clients	198
Microsoft SQL Server		server-initiated synchronization	137
as MobiLink consolidated database	14	starting	16
stored procedure calls	66	stopping the MobiLink server	17
ML directive		synchronization logic	38
Redirector	318	synchronization techniques	69, 414, 429
ML_CLIENT_TIMEOUT directive		transport-layer security	338
Redirector	318	Tutorial - Using Adaptive Server	
ml_user		Anywhere	369
installing a remote database over an		Tutorial - Using an Oracle database	401
old one	170	Tutorial - Using MobiLink sample	
ml_username		applications	413, 429
about	20	Tutorial - Using Sybase Central	383
script parameter	48	UltraLite clients	207
mlDomConfig.xml		uploading rows	54
about	256	MobiLink .NET API reference	269
mlMonitorSettings		MobiLink connections	
MobiLink Monitor settings	305	debugging	336
mlscript.jar		MobiLink consolidated databases	
MobiLink Java synchronization logic	229	Adaptive Server Anywhere as	12
MLStartClasses		Adaptive Server Enterprise as	12
.NET user-defined start classes	263	IBM DB2 as	13
mlxtract utility		Oracle as	13
sp_hook_dbxtract_begin procedure	84	SQL Server as	14
MobiLink		MobiLink data types	
.NET synchronization logic	251	.NET and SQL	261
a simple synchronization script	39	Java and SQL	233
Adaptive Server Anywhere clients	167	MobiLink download stream	
architecture	8	defined	21
configuring web servers	314	MobiLink Java API reference	246
database connections	293	MobiLink Monitor	
deprecated features from version 7	200	about	298
development tips	71	Chart pane	304
events during download	56	description of user interface	302
features	4	Details Table pane	303
fundamental rules	71	Options	305
isolation levels	21	Overview pane	305
Java synchronization logic	227	Properties	306
key factors	290	restoring defaults	305
options for writing synchronization		saving data	307
logic	31	specifying watches	308
performance	285	starting	299
process overview	21	statistical properties	310
running	329	using	302
sample application	414	viewing in MS Excel	307
scheduling Adaptive Server Anywhere		Watch Manager	308

MobiLink performance		about	16
about	285	HotSync	211
key factors	290	multiple instances	318
MobiLink security		starting	16
changing passwords	113	stopping	16
choosing a user authentication		tutorial	375
mechanism	107	MobiLink synchronization subscriptions	
custom user authentication	114	about	182
new users	111	MobiLink system tables	
passwords	110	creating in consolidated database	11
user authentication	103	MobiLink upload stream	
user authentication architecture	108	defined	21
user authentication passwords	112	processing	27
MobiLink server		MobiLink user creation wizard	
troubleshooting startup	346	using	178
MobiLink synchronization		MobiLink user name	20
Adaptive Server Anywhere clients	167	script parameter	48
clients	19	MobiLink user names	
custdb sample database	429	about	104
file-based downloads	117	Contact sample	420
fundamental rules	71	CustDB sample	443
scheduling Adaptive Server Anywhere		MobiLink users	
clients	198	about	103
server-initiated synchronization	137	adding to a remote database	178
UltraLite clients	207	configuring properties at a remote	
writing .NET classes	262	database	179
writing Java classes	234	creating	104
MobiLink synchronization client		creating in remote databases	178
tutorial	377	dropping from a remote database	180
MobiLink synchronization logic		passwords	179
.NET and SQL data types	261	sharing a name	105
data types for .NET and SQL	261	modems	
data types for Java and SQL	233	Palm Computing Platform	216
Java and SQL data types	233	monitor	
MobiLink synchronization scripts		MobiLink Monitor	297
constructing .NET classes	261	monitoring	
constructing Java classes	233	synchronizations in MobiLink	297
database transactions and .NET classes	261	multiple applications	
database transactions and Java classes	232	differentiating MobiLink scripts	49
debugging Java classes	235	N	
preserving database transactions	232, 261	Netscape web servers	
writing .NET classes	262	configuring the NSAPI Redirector	320
writing Java classes	234	new users	
MobiLink synchronization server		MobiLink user authentication	111
		newsgroups	
		technical support	xvi

- Notifier properties file
 - about 145
- Notifiers
 - about 138
 - configuring 145, 147
 - properties 145
 - starting 145
- NSAPI Redirector
 - configuring 320
- NUMERIC data type
 - MobiLink and Adaptive Server Enterprise 67
- O**
- objects
 - MobiLink .NET API 269
 - MobiLink Java API 246
- ODBC
 - multiple errors 63
- ODBC data sources
 - for MobiLink synchronization 12
- options
 - priority order for MobiLink extended options 180
- options for performance tuning
 - MobiLink 186
- options for writing synchronization logic
 - 31
- Oracle
 - as MobiLink consolidated database 13
 - MobiLink tutorial 401
 - packages in MobiLink synchronization 65
 - sequences in MobiLink synchronization 65
- overlaps
 - partitioning 77
- P**
- packages
 - session-wide information 65
- Palm Computing Platform
 - HotSync synchronization 211
 - synchronization 216
 - TCP/IP synchronization 216
- Palm devices
 - Listener 160
- parameters
 - last download timestamp 48
 - ml_username 48
 - MobiLink table name 48
 - MobiLink user name 48
 - synchronization scripts 48
- partitioning
 - column-wise 172
 - data among MobiLink remote databases 169
 - defined 77
 - disjoint 77
 - row-wise 173
- partitioning rows
 - Contact sample 421, 423
- partitioning tables
 - example 77
- parts of the synchronization system 8
- passwords
 - changing for MobiLink 113
 - MobiLink user authentication 110, 112
 - MobiLink users 179
- performance
 - downloads 289
 - MobiLink 285
 - MobiLink upload stream processing 27
- performance tips
 - MobiLink 286
- Personal web Manager
 - configuring 323
- port stream parameter
 - synchronizing across firewalls 316
- preparing
 - remote databases for MobiLink 169
- primary key pools
 - example 87
 - generating unique values using default global autoincrement 82
 - synchronization 86
- primary keys
 - MobiLink and Adaptive Server Enterprise 67
 - Oracle sequences 65
 - primary key pools 87
 - uniqueness in synchronization 81
- priority order for extended options and connection parameters 180

- priority synchronization
 - MobiLink performance 289
- private assemblies
 - implementing in MobiLink 255
- procedural language
 - role of in MobiLink synchronization 24
- properties
 - Notifier 145
 - server-initiated synchronization 145
- protocols
 - MobiLink synchronization 8
- public key cryptography
 - about 338
- publication creation wizard
 - column-wise partitioning 172
 - creating MobiLink publications 171
 - row-wise partitioning 174
- publications
 - altering 175
 - column-wise partitioning 172
 - creating 171
 - dropping 177
 - row-wise partitioning 173
 - simple 171
 - using a WHERE clause 173
- publishing
 - selected columns 172
 - selected rows 173
 - tables 171
 - whole tables 171
- publishing data 171
- publishing only some columns in a table 172
- publishing only some rows in a table 173
- publishing whole tables 171
- push requests
 - about 138, 143
- push technology
 - server-initiated synchronization 137
- R**
- RAS
 - about 216
 - configuring 219
- recording errors during synchronization 62
- Redirector
 - about 313
 - configuring (all versions) 318
 - configuring for servlet version 325
 - configuring the ISAPI version for
 - Microsoft web servers 323
 - configuring the NSAPI version 320
 - configuring the servlet Redirector for
 - Tomcat 325
 - MobiLink requests 314
 - specifying the location 318
- redirector.config
 - configuring 318
 - location 318
- REDIRECTOR_HOST directive
 - Redirector 318
- REDIRECTOR_PORT directive
 - Redirector 318
- referential integrity
 - during MobiLink synchronization 28
- registering
 - MobiLink Adaptive Server Anywhere
 - applications with ActiveSync 192
 - MobiLink UltraLite applications with
 - ActiveSync 224
- registry
 - HotSync parameters 211
- Remote Access Service
 - about 216
 - configuring 219
- remote databases
 - creating Adaptive Server Anywhere
 - clients 168
 - deploying Adaptive Server Anywhere
 - databases 168
 - SQL scripts 169
- remote DBA permissions
 - MobiLink synchronization 185
- remote MobiLink databases
 - schema changes 100
- remote tables
 - deleting rows in MobiLink 58
- removing
 - articles 175
- replication
 - MobiLink synchronization
 - subscriptions 182

report_error		Contact MobiLink sample errors	428
syntax	62	Contact MobiLink sample Product	
reporting errors during synchronization	62	table	425
reqtool		Contact MobiLink sample SalesRep	
how to use	360	table	421
requests		Contact MobiLink sample statistics	
routing	314	428	
requirements		Contact MobiLink sample tables	417
MobiLink consolidated databases	10	Contact MobiLink sample users	420
resolution		CustDB MobiLink sample tables	440
MobiLink conflict resolution	90	CustDB MobiLink sample	
resolve_conflict		ULCustomer table	446
Contact sample	426	CustDB MobiLink sample ULOrder	
resolving		table	444
MobiLink conflicts	90	CustDB MobiLink sample ULProduct	
return values		table	447
.NET synchronization	262	CustDB MobiLink sample users	443
Java synchronization	234	Java synchronization logic	240
reverse proxy		scheduling	
defined	314	MobiLink Adaptive Server Anywhere	
role of digital certificates	342	clients	198
routing requests		MobiLink server-initiated	
MobiLink synchronization	314	synchronization	159
rows		schema	
partitioning	77	custdb sample database	430
rsaserver.crt	346	schema changes	
running .NET synchronization logic	255	remote MobiLink databases	100
running outside the current session		schemas	
MobiLink	329	relating consolidated tables to	
		MobiLink remote tables	11
S		script parameters	
sample application		about	48
MobiLink CustDB application	429	script types	46
MobiLink database schema	432	script versions	
sample database		adding	50
MobiLink CustDB application	429	configuring at remote databases	179
MobiLink database schema	432	in MobiLink synchronization	49
sample domain configuration file		scripts	
about	256	about MobiLink	22
sample.crt	346	adding to the consolidated database	51
samples		automating MobiLink synchronization	
.NET synchronization logic	266	40	
Contact MobiLink sample	414	common parameters	48
Contact MobiLink sample Contact		connection scripts	46
table	423	MobiLink synchronization	16
Contact MobiLink sample Customer		supported DBMS scripting strategies	
table	421	65	

table scripts	46	dependencies	334
versions	49	removing	331
writing scripts to download rows	56	running MobiLink	329
writing scripts to upload rows	54	running multiple	334
scripts and the synchronization process	44	Windows	331
secure socket layers		servlet Redirector	
obtaining certificates	360	Apache Tomcat	325
with MobiLink synchronization	338	configuring	325
security		servlets	
changing MobiLink passwords	113	installing	325
MobiLink	338	session-wide variables	
MobiLink client architecture	340	IBM DB2 in MobiLink	
MobiLink custom user authentication		synchronization	66
114		Oracle packages	65
MobiLink synchronization	185	setup	
MobiLink user authentication	103, 107	MobiLink .NET synchronization logic	
new MobiLink users	111	253	
user authentication passwords	112	MobiLink Java synchronization logic	
self-signed certificates	347	229	
making	348	shared assemblies	
using	349	implementing in MobiLink	255
sequences		ShutdownCallback	
primary key uniqueness in MobiLink		MobiLink .NET API	279
synchronization	65	ShutdownListener	
server authentication		MobiLink Java API	250
MobiLink	344	simple synchronization script	39
server-initiated synchronization		SLEEP directive	
about	137	Redirector	318
automatic connection recovery	148	SMTP gateway	
configuring and starting the Listener		Notifier properties	151
154		snapshot synchronization	
Listener SDK	162	about	74
sample walkthrough	164	Contact sample	421
supported platforms	141	example	75
unguaranteed delivery	163	sp_hook_dbxtract_begin procedure	
ServerContext		unique primary keys	84
MobiLink .NET API	277	using	84
MobiLink Java API	248	SQL Anywhere Studio	
ServerException		documentation	x
MobiLink .NET API	279	SQL Server	
MobiLink Java API	250	as MobiLink consolidated database	14
servers		SQL synchronization logic	
about MobiLink synchronization	16	alternatives	31
service dependencies		MobiLink	38
MobiLink	334	SqlHook.beginPoll	148
services		SqlHook.endPoll	151
configuring	331	SQLType	

MobiLink .NET API	280	syncase.sql	
start classes		about	12
.NET synchronization logic	263	syncase125.sql	
Java synchronization logic	237	about	12
starting		syncdb2.sql	
MobiLink synchronization from		about	13
UltraLite applications	19	synchronization	
MobiLink synchronization server	16	about MobiLink	7, 337
statement-based scripts		ActiveSync for MobiLink Adaptive	
uploading rows	54	Server Anywhere clients	189
statement-based uploads		architecture of the MobiLink system	8
conflict detection	90	changing passwords	113
StaticCursorLongColBuffLen		conflict resolution	90
Adaptive Server Enterprise	66	custom user authentication	114
statistical properties		deleting rows	58
MobiLink	310	downloading rows	56
stop		HotSync Palm Computing Platform	
MobiLink synchronization server	17	211	
STOP SYNCHRONIZATION DELETE		initiating	185
statement		many-to-many relationships	78
Adaptive Server Anywhere clients	193	MobiLink performance	27
stopping		MobiLink process overview	21
MobiLink synchronization server	16,	MobiLink scripts	16
17		MobiLink synchronization server	
upload of deletes using MobiLink	193	authentication	346
stored procedures		MobiLink transactions	25
calling in MobiLink synchronization		MobiLink tutorial	369, 383, 401
using ODBC syntax	66	ODBC data sources for MobiLink	12
MobiLink stored procedure source		options for writing synchronization	
code	53	logic	31
using to add or delete synchronization		Palm Computing Platform	216
scripts	52	performance tips	285
using to download data	97	process	44
storing user name during conflict		running the MobiLink synchronization	
resolution	93	server	329
stream_parms synchronization parameter		scheduling MobiLink Adaptive Server	
HotSync conduit	214	Anywhere clients	198
HotSync synchronization	210	server-initiated	137
subscribing MobiLink synchronization		snapshot	75
users	182	techniques	69
subscriptions		timestamps in MobiLink	25
MobiLink synchronization	182	transport-layer security with MobiLink	
support		338	
newsgroups	xvi	writing MobiLink scripts in .NET	251
supported DBMS scripting strategies	65	writing MobiLink scripts in Java	227
syncasa.sql		writing scripts	37
about	12	synchronization basics	7

synchronization clients		synchronization server	
Adaptive Server Anywhere or		about MobiLink	16
UltraLite for MobiLink	19	synchronization subscriptions	
synchronization conduit		altering	183
HotSync	217	dropping	183
synchronization definitions		MobiLink	182
differences from version 7	200	options	180
rewriting version 7	204	synchronization techniques	
writing	202	custdb sample application	429
synchronization errors		data entry	94
handling MobiLink	62	deleting rows	95
synchronization event hook sequence	194	failed downloads	96
synchronization logic		MobiLink tutorial	413
MobiLink	38	partitioning	77
options for writing	31	primary key pools	86
synchronization scripts		snapshot-based synchronization	74
.NET	251	stored procedures to download	97
.NET methods	262	timestamp-based synchronization	72
about	38	uploading rows	54
adding and deleting	51	synchronization upload stream	
adding or deleting with stored		MobiLink processing	27
procedures	52	synchronization user names	
adding with Sybase Central	51	MobiLink	104
automatic generation	40	synchronization users	
common parameters	48	about	103
connection scripts	46	adding to a remote database	178
DBMS dependencies	65	configuring properties at a remote	
download_cursor	56	database	179
example	39	creating	104
example generation	41	creating in remote databases	178
examples	43	dropping from a remote database	180
execution during	44	multiple	185
handle_error event	62	sharing a name	105
implementing for .NET	253	SynchronizationException	
implementing for Java	229	MobiLink .NET API	282
Java	227	MobiLink Java API	250
Java methods	234	synchronizing	
report_error	62	databases with MobiLink	337
supported DBMS scripting strategies		syncmss.sql	
65		about	14
table scripts	46	syncora.sql	
testing	64	about	13
types	46	using	405
versions in MobiLink	49	system tables	
writing	37	creating in MobiLink consolidated	
writing scripts to download rows	56	database	11
writing scripts to upload rows	54		

T

table
 script parameter 48
table scripts
 about 46
 adding with Sybase Central 51
 defined 39, 46
tables
 adding to remote MobiLink databases 100
 column-wise partitioning 172
 partitioning 77
 publishing 171
 relating consolidated tables to
 MobiLink remote tables 11
 row-wise partitioning 173
TCP/IP synchronization
 Palm Computing Platform 216
technical support
 newsgroups xvi
template.notifier
 about 145
temporarily stopping synchronization of
 deletes 193
testing
 synchronization scripts 64
testing script syntax 64
timestamp-based synchronization
 about 72
 Contact sample 421, 423
 download_cursor script 73
 download_delete_cursor script 72
tips
 synchronization techniques 71
Tomcat
 configuring the servlet Redirector 325
 supported versions 325
transaction log
 location for dbmlsync 187
transactions
 during MobiLink synchronization 25
 in MobiLink synchronization scripts 232, 261
transport-layer security
 about 337
 invoking 346
 MobiLink 338

 MobiLink client architecture 340
 obtaining certificates 360
troubleshooting
 conduit 213
 dial-up networking 221
 handling failed downloads 96
 HotSync conduit 215
 MobiLink 336
 MobiLink deployment 170
 MobiLink security 346
 MobiLink synchronization server log 17
 RAS 221
 Remote Access Service 221
tutorials
 MobiLink 383
 MobiLink custdb sample applications 429
 MobiLink sample applications 413
 MobiLink with Adaptive Server
 Anywhere clients 369
 MobiLink with Oracle 401
 MobiLink with Sybase Central 383

U

UDB
 as MobiLink consolidated database 13
UDP gateway
 Notifier properties 152
UdpGateway.sender 153
UL_DEBUG_CONDUIT environment
 variable
 troubleshooting conduit 213
UL_DEBUG_CONDUIT_LOG
 environment variable
 troubleshooting HotSync conduit 215
ULPalmExit function
 using 211
ULSynchronize library function 19
UltraLite 43
 MobiLink clients 19
UltraLite applications
 as MobiLink clients 19
UltraLite clients
 MobiLink 19, 207
unique primary keys

generating using global autoincrement		writing scripts	54
82		url_suffix stream parameter	
generating using key pools	86	synchronizing across firewalls	316
generating using UUIDs	81	USB	
MobiLink installations	81	HotSync support for	209
Universal Serial Bus		user authentication	
HotSync support for	209	.NET synchronization logic	114
unknown_timeout stream parameter		changing MobiLink passwords	113
synchronizing across firewalls	316	choosing a mechanism in MobiLink	
upgrading		107	
schemas in MobiLink remote		Java synchronization logic	114
databases	100	MobiLink architecture	108
upgrading applications		MobiLink custom mechanism	114
using multiple MobiLink script		MobiLink passwords	110
versions	49	MobiLink security	103
upload cache size		new MobiLink users	111
MobiLink performance	287	passwords	112
upload events		user names	
about	54	MobiLink	104
upload stream		MobiLink client names	20
defined	21	user-defined start classes	
events	54	MobiLink .NET	263
MobiLink transactions	25	MobiLink Java	237
processing of MobiLink	27	user-specific conflict resolution	93
upload-only synchronization		users	
about	30	about MobiLink	104
upload_delete		Using a global certificate as a server	
Contact sample	423, 425	certificate	361
CustDB sample	447	using a globally-signed certificate as an	
upload_fetch		enterprise certificate	364
conflict detection	90	using a self-signed certificate	349
Contact sample	426	using ActiveSync synchronization	
upload_insert		MobiLink Adaptive Server Anywhere	
Contact sample	422, 424	clients	189
CustDB sample	446	using stored procedures to add or delete	
upload_new_row_insert		synchronization scripts	52
Contact sample	426	using the signed certificates	356
storing user name	93	UUIDs	
upload_old_row_insert		MobiLink synchronization application	
Contact sample	426	81	
storing user name	93		
upload_update		V	
conflict detection	90	VARCHAR data type	
Contact sample	423, 425, 426	MobiLink and other DBMSs	67
CustDB sample	447	verifying certificate fields	363
uploading rows		verifying fields in certificate chains	364
MobiLink performance	289	versions	

adding script versions	50	writing upload_update scripts	54
of MobiLink synchronization scripts	49		
Visual Basic			
support in MobiLink .NET	252		
W			
web servers			
configuring	314		
configuring for synchronization	318		
configuring ISAPI Microsoft for synchronization	323		
configuring NSAPI Netscape for synchronization	320		
MobiLink clients and	316		
WHERE clause			
publications	173		
wizards			
add connection script	51		
add service	331		
add synchronized table	51		
add synchronizing table script	52, 389		
add user	110		
add version	50, 389		
article creation	175		
create database	385		
MobiLink user creation	178		
publication creation	171		
worker threads			
MobiLink	290		
MobiLink performance	286		
writing			
.NET synchronization logic	251		
Java synchronization logic	227		
writing download_cursor scripts	56		
writing download_delete_cursor scripts	58		
writing scripts to download rows	56		
writing scripts to handle errors	62		
writing scripts to upload rows	54		
writing SQL synchronization scripts	37		
writing synchronization scripts			
SQL	37		
supported DBMS scripting strategies	65		
writing upload_delete scripts	55		
writing upload_fetch scripts	55		
writing upload_insert scripts	54		